

AD-A126 024

A SURVEY OF SOME APPROACHES TO DISTRIBUTED DATA BASE &
DISTRIBUTED FILE SYSTEM ARCHITECTURE(U) BGS SYSTEMS INC
WALTHAM MA P 5 MAGER ET AL. JAN 80 TR-80-001

1/1

UNCLASSIFIED

N00140-79-C-8933

F/G 9/2

NL

END

FILMED

21

DTIC

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

3

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER TR-80-001	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Survey of Some Approaches to Distributed Data Base & Distributed File System Architecture		5. TYPE OF REPORT & PERIOD COVERED
7. AUTHOR(s) Peter S. Mager Robert P. Goldberg		6. PERFORMING ORG. REPORT NUMBER TR-80-001
9. PERFORMING ORGANIZATION NAME AND ADDRESS BGS Systems, Inc. Waltham, MA 02154		8. CONTRACT OR GRANT NUMBER(s)
11. CONTROLLING OFFICE NAME AND ADDRESS NUSC - New London New London, CT		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		12. REPORT DATE January, 1980
		13. NUMBER OF PAGES
		15. SECURITY CLASS (if this report) Unclassified
16. DISTRIBUTION STATEMENT (of this Report) Unlimited		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Distributed Systems, Distributed Data Bases, Data Base Management, File Systems Architecture		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) This report gives an overview of 7 approaches to distributed data sharing as exemplified by Cullinane's CDMS, Xerox PARC's experimental systems, Tandem's Guardian/Expand/Enscribe, CCA's SDD-1, UC Berkeley's Distributed INGRES & MUFFIN and the University of Wisconsin's DIRECT data base machine. It lays the ground work for an architectural reference model for distributed data sharing systems that is described in a subsequent report.		

DTIC
ELECTE
S **D**
MAR 24 1983
E

AD A 126024

DTIC FILE COPY

**A Survey of Some Approaches to
Distributed Data Base and
Distributed File System Architecture**

Prepared by:

Peter S. Mager

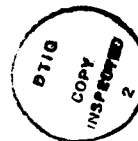
Robert P. Goldberg

BGS Systems, Inc.

Waltham, Mass.

TR-80-001

January 1980



Accession For	
NTIS GRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Avail and/or	
Dist	Special
A	

83 03 24 045

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER TR-80-001	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Survey of Some Approaches to Distributed Data Base & Distributed File System Architecture		5. TYPE OF REPORT & PERIOD COVERED
7. AUTHOR(s) Peter S. Mager Robert P. Goldberg		6. PERFORMING ORG. REPORT NUMBER TR-80-001
9. PERFORMING ORGANIZATION NAME AND ADDRESS BGS Systems, Inc. Waltham, MA 02154		8. CONTRACT OR GRANT NUMBER(s)
11. CONTROLLING OFFICE NAME AND ADDRESS NUSC - New London New London, CT		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		12. REPORT DATE January, 1980
		13. NUMBER OF PAGES
		15. SECURITY CLASS. (if this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Unlimited		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Distributed Systems, Distributed Data Bases, Data Base Management, File Systems Architecture		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) This report gives an overview of 7 approaches to distributed data sharing as exemplified by Cullinane's CDMS, Xerox PARC's experimental systems, Tandem's Guardian/Expand/Enscribe, CCA's SDD-1, UC Berkeley's Distributed INGRES & MUFFIN and the University of Wisconsin's DIRECT data base machine. It lays the ground work for an architectural reference model for distributed data sharing systems that is described in a subsequent report.		

Table of Contents

Foreword	1
1 Introduction	3
2 Cullinane CDMS	7
3 Xerox PARC style Distributed Processing	11
4 Tandem GUARDIAN/EXPAND/ENSCRIBE	19
5 SDD-1	27
5.1 Overall Architecture	28
5.2 Logical Structure	30
5.3 Run Time Scenario	32
5.4 Methods and Algorithms	33
5.4.1 The concurrent update problem	33
5.4.2 Distributed Query Processing	34
5.4.3 Reliable Writing	35
6 Distributed INGRES	37
7 MUFFIN	43
8 DIRECT	49
9 Summary	55
10 References	63
Index	67

List of Figures

Figure 2-1:	CDMS data flow	9
Figure 3-1:	Typical DFS Configuration	12
Figure 3-2:	Paxton's Transaction Primitives	14
Figure 3-3:	Paxton's Transaction Primitives, continued	15
Figure 4-1:	The Tandem NonStop TM System Architecture	22
Figure 4-2:	Mirrored Disc Volumes on a Tandem System	23
Figure 4-3:	Logical Organization of the GUARDIAN Operating System	24
Figure 4-4:	A Typical ENSCRIBE file configuration	25
Figure 5-1:	Decomposition of SDD-1 into Local and Global Parts	29
Figure 5-2:	SDD-1 Logical Architecture	31
Figure 6-1:	Logical Organization of Distributed INGRES	38
Figure 6-2:	Distributed INGRES Logical Interactions	40
Figure 7-1:	MUFFIN logical architecture	45
Figure 7-2:	D-cell Physical Layout	46
Figure 7-3:	A-cell Run-Time Environment	47
Figure 7-4:	The MUFFIN Hardware	48
Figure 8-1:	DIRECT system architecture	50
Figure 8-2:	DIRECT tree structured query packet	51
Figure 8-3:	Proposed DIRECT Physical Configuration	53
Figure 9-1:	The Network Environment of Some Distributed Systems	56
Figure 9-2:	Coupling between Logical Components in Some Distributed Systems	58
Figure 9-3:	Division of DBMS Functionality among System Components	59
Figure 9-4:	Comparison of Functional Issues	60
Figure 9-5:	Comparison of Functional Issues, Continued	61


Foreword

This document presents a survey of current approaches to building distributed data base and file systems. The study was sponsored by the Naval Underwater Systems Center, New London, Connecticut under contract 7/L N00140-79-C-8933. The study was performed and report written by Peter S. Mager and Robert P. Goldberg. The technical monitor for this study was James P. Shores.

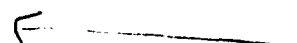
This is an interim report in a continuing study of the characteristics of distributed data base management systems.

January 1980

1 Introduction



This report looks at seven existing systems that illustrate a variety of approaches toward distributed data base and file management. The emphasis is on architectural design and the overall logical structure of the systems. Algorithms and unusual features are discussed where they are thought to be particularly interesting or to have an important effect on the overall system structure.

The goal is to illustrate architectural tradeoffs and identify issues that may arise if a distributed data base system is incorporated into real-time tactical or process control systems. A secondary goal is to identify the range of architectures and system designs that must be provided for in a formal reference model that would be useful in the design of future Distributed Data Base Management Systems (DDBMSs). 

In this context we have chosen seven systems that are representative of the technology currently available or under development. The next three sections (chapters 2 - 4) describe working systems implemented by three vendors (Cullinane, Xerox, and Tandem); they illustrate alternative approaches toward dividing data base management functionality among nodes in a network. Cullinane puts most of its functionality in a rough analog of a back-end data base machine; Xerox puts it in front-end (what it calls "client") machines; and Tandem divides it in a dynamic manner making use of the special characteristics of its architecture. The fifth chapter describes SDD-1, a prototype system having full DDBMS functionality, and discusses some of the problems and special considerations that must be addressed to implement that type of system. The last three survey chapters (chapters 6 through 8) illustrate how a single DBMS (INGRES) can be tailored in three different ways to a general geographically distributed, local area and multi-processor environment.

Cullinane's CDMS represents perhaps the simplest approach to distributed management of data. In this system conventional commercial data base management systems are located on multiple host computers within a network. Application programs on other hosts can access this data

through special communication/data base front-end software running on their local machine. This software makes the remote application look like a local program to the DBMS it accesses. However, the application program or end user must know the location (name of the host computer) at which the data is stored. In addition, no system facilities are provided to keep redundant data on multiple machines consistent or to guarantee the integrity (atomicness) of transactions that update or access data on multiple machines. In practice this means that application programs generally access data on only one machine at a time.

The Xerox PARC systems illustrate a somewhat different way of supporting remote access to data. Their approach is specifically oriented to use in an environment of small computers linked together by a high band width local network. Like Cullinane's CDMS, they divide data base management into front-end and back-end parts. However, in contrast to Cullinane which puts most of its data base functionality in its back-end data base managers, the Xerox systems put functionality associated with data structuring and formatting in the user controlled front-end processors. Data transferred across the network consists of blocks of uninterpreted byte streams; the front-ends control data base management, making use of what looks to them like intelligent remote disks. The most advanced of the Xerox systems, DFS, enhances this by providing mechanisms for increased reliability and for insuring consistency of transactions that access multiple back-end servers; these mechanisms are described in Chapter 3.

The TANDEM computer system illustrates how redundant copies of hardware and software components can be connected to provide very high reliability. Most of the support facilities for this redundancy are provided at the hardware and operating system level.

The techniques used illustrate how the underlying support environment can play a large role in determining the effectiveness and efficiency of distributed data management facilities implemented on top of them.

However, the commercial TANDEM product provides only rudimentary functionality for replicating data at a logical level. Discs are replicated in their entirety; transaction management and concurrency

control are mostly under user or application program control.

SDD-1 represents the opposite extreme from TANDEM. It is an experimental general purpose, high functionality distributed data base management system implemented to run on a general purpose, geographically dispersed network (the ARPANET) where it may not be feasible to provide special functionality to enhance DDBMS performance or reliability within the network itself. Because of this an additional protocol layer (called the Reliable Network) is implemented on top of the normal network functions as part of the DDBMS system. In addition, the logical structure of the DDBMS is partitioned to allow global and local data management functions to be handled separately. The issues involved in doing this are discussed in Chapter 5. Some of the more important algorithms and special techniques used to support this are also described briefly.

The last three chapters describe research approaches to distributing the functionality of INGRES, a relational data base management system developed at the University of California, Berkeley. They illustrate how the same basic system can be implemented using a spectrum of implementation techniques. Distributed INGRES is an attempt to provide distributed data base management functionality by partitioning control among primary sites, each of which controls access to its local data and redundant copies of that data at other sites. The consequences of this approach are discussed in Chapter 6. MUFFIN is a specialized attempt to adapt distributed INGRES to a local area network environment. Functionality is divided between application and data management virtual machines. MUFFIN is discussed in Chapter 7. DIRECT can be thought of as one way of building a data base machine to support INGRES using multiple processors tied together by shared memory. Its unusual architecture is discussed in Chapter 8.

These seven systems illustrate the diversity of approaches that are being used to support shared distributed data bases in commercial, experimental and research environments. They represent some of the choices that are available for potential Navy real time process control distributed systems. The chapters that follow describe these systems with a view to highlighting the features that might be of particular usefulness in potential Navy systems as well as the issues that would have to be addressed in incorporating distributed data base functionality into complex

January 1980

real time systems.

2 Cullinane CDMS

The Cullinane Corporation of Wellesley, MA markets a "distributed" data management system CDMS (Cullinane Data Management System) based around its IDMS data base management system and its Integrated Data Dictionary. The DBMS is based on a CODASYL compatible network type data model.¹ It was designed to run on large IBM mainframes, but parts of the system (e.g. data base management) have been implemented for minicomputers (e.g. DEC PDP-11's).

In the CDMS architecture (see Figure 2-1) application programs run in a separate partition from the data base management system and communicate with the data base manager via subroutine calls to a small application program/database interface that runs in the same partition as the application program. The interface routine then forwards the database transactions to the DBMS using a message passing type of protocol.

This division of labor facilitates the generalization of allowing the application program with its front-end interface to reside on a separate host computer from the data base manager. When the application program is on a different host computer from the DBMS, an intermediate facility (running on the same computer as the application program) called multiple computer support takes the transactions generated by the application program/database interface routine and routes them to the appropriate computer using the available communication line handlers.

Multiple front-ends can link to a common back-end DBMS (as in Figure 2-1). There can be multiple back-end DBMS's. However database partitions

¹The network data model is the type of data structure supported on a single node; it does not imply anything about communications networking facilities.

(in the case of IDMS AREAS)² cannot overlap and redundancy between database fragments controlled by different data base managers is not supported. In particular no mechanisms are provided to support cross partition locking or update synchronization at the system level.

The application programs are responsible for knowing which data base partition to access and for handling any cross partition relationships; no location transparency is provided.

²AREAS is a functional way of partitioning data bases that is supported in the Codasyl data model

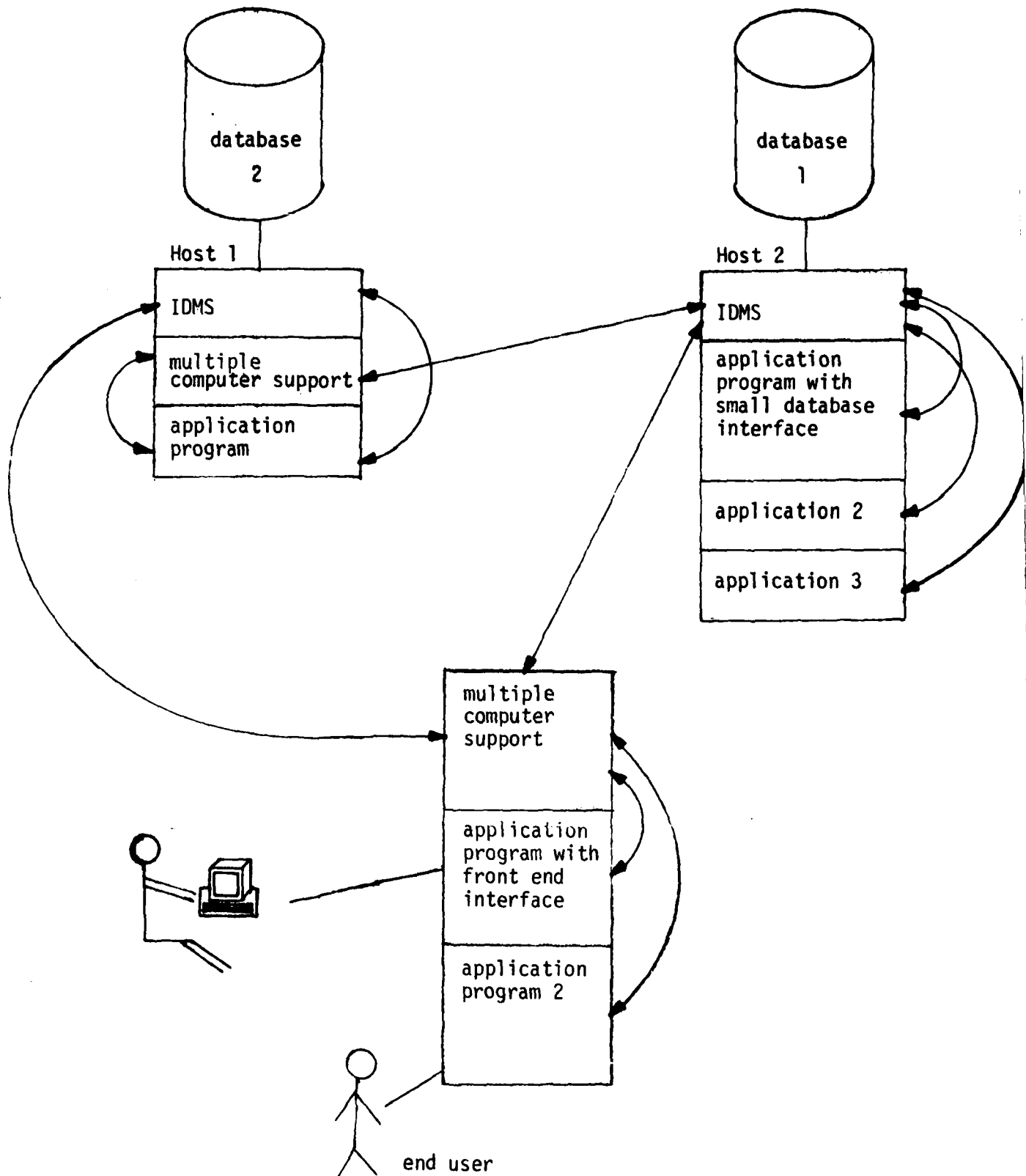


Figure 2-1: CDMS data flow

January 1980

3 Xerox PARC style Distributed Processing

Xerox PARC (Palo Alto Research Center) researchers have published three papers [Swinehart 79], [Israel 78], [Paxton 79] describing research into a client-server approach to file sharing over a local area contention based network (the original Ethernet [Metcalfe, R. M., and Boggs, D.R. 76]).

In the Xerox system personal computers (e.g. Altos) running application programs (clients) access files managed by servers running on back-end processors. An intermediate facility, the Distributed File System (DFS), running on the server machines, coordinates transactions that span several servers (on different processors) and provides some crash recovery facilities. Servers supply data a page (or linear sequence of bytes) at a time. Clients handle all higher level data structuring and transformations. DFS handles multi-server synchronization and provides some common code for communication and what appears to the application programs as remote disk access.

A typical DFS configuration is shown in Figure 3-1. In the figure, three client machines access three servers over an Ethernet. The client programs are responsible for all interpretation and manipulation of data. The servers act as highly intelligent disk controllers. DFS oriented common code in the client "personal computers" provide some common functionality for communications and other system tasks.

The Xerox papers describe three variations on the client server model.

In the WFS system of Swinehart McDaniel and Boggs [Swinehart 79], the server machines are independent and act as remote disks satisfying client machine requests a disk page at a time. Each disk page access involves an independent request and the client programs are responsible for providing such traditional file system facilities as stream I/O and a directory system. This division of facilities eliminates the need for WFS to maintain much transitory state information between page requests, but a timeout lock is provided.

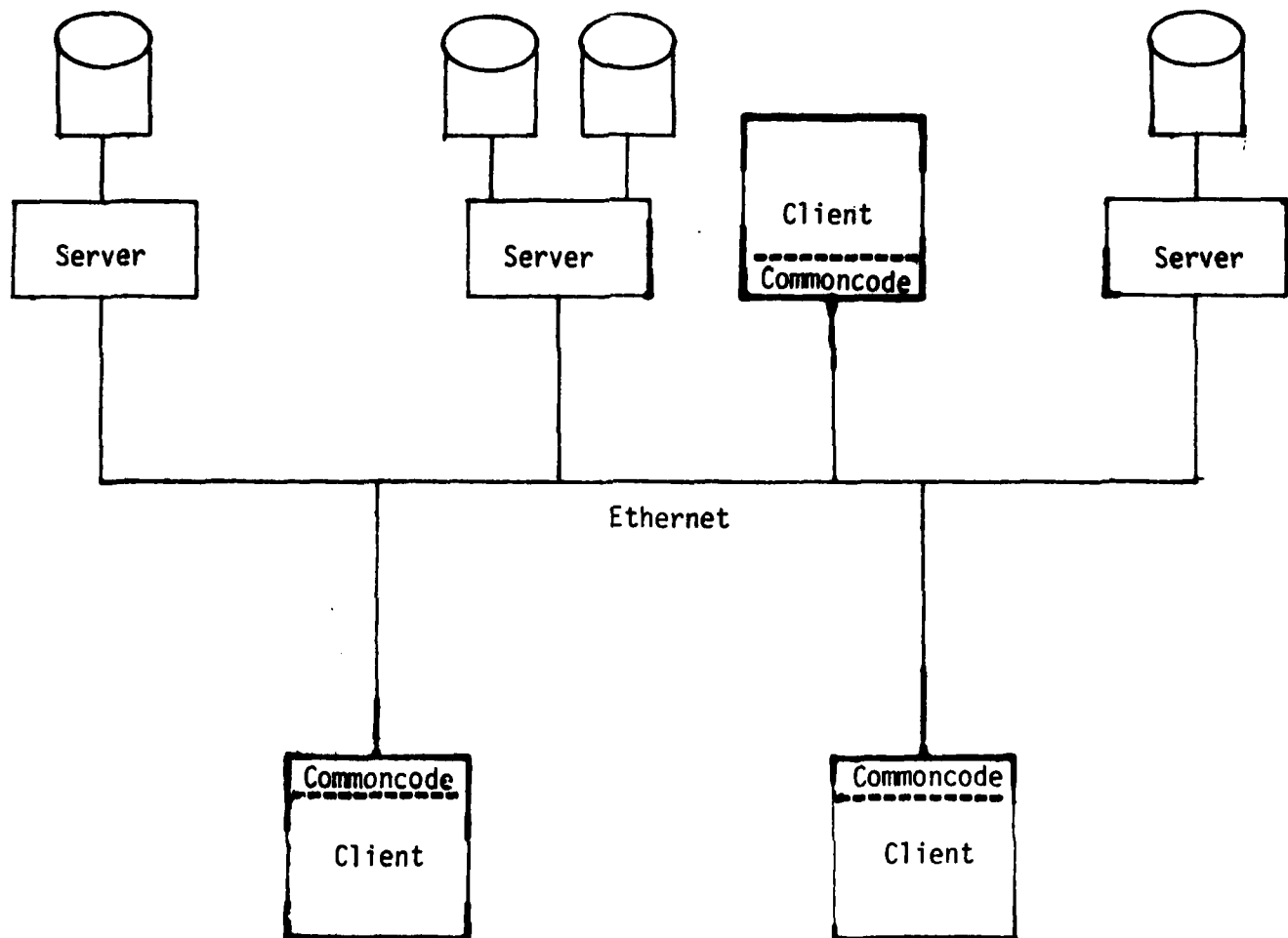


Figure 3-1: Typical DFS Configuration

The WFS primitive operations include:

- o reading and writing pages of files,
- o allocating and deallocating file identifiers (FIDs) and disk file pages,
- o assigning and modifying file properties, and
- o performing system maintenance activities including those needed for recovery after a system crash.

Paxton [Paxton 79] describes a transaction oriented system that runs on a client machine and interfaces to one or more WFS file servers. The transaction system is designed to ensure the consistency and atomicness of transactions. Consistency guarantees that each transaction will get a view of shared data that is independent of any other transactions that are running concurrently (Some other authors refer to this as serializability.) Atomicness guarantees that for each transaction either all writes will be executed or none will, independent of crashes in servers or clients.

To accomplish this, Paxton's system uses intentions files that contain information about data files changed by a transaction. These intentions files can be used in crash recovery. Paxton introduces six basic actions that can be performed by client programs as part of a transaction:

- o Begin Transaction
- o Open
- o Read
- o Write
- o Abort Transaction
- o End Transaction

A description of these is given in Figure 3-2.

The Distributed File System described by Israel, Mitchell and Sturgis [Israel 78] extends the WFS facility to provide some of the features needed for multiple server synchronization. In DFS one server is chosen as

BEGIN TRANSACTION

This routine is called at the beginning of each transaction to change the state to **STARTED**. The following routines signal an error if they are called in any other state.

OPEN

OPEN receives a file identifier and returns a handle for use in subsequent accesses. The identifier includes data indicating where the file is located, so the following operations can communicate with the appropriate server. **OPEN** locks the file, saves the key, and reads the header. If the header indicates that the file is involved in another transaction, a crash must have previously occurred preventing the completion of that transaction, so recovery is done in a manner described below and the header is then reread.

Note that the header is not written at this time; it is modified only in **END TRANSACTION**, and then only if there have been **WRITE**'s to the file.

READ

The arguments to **READ** are a file handle, a logical sector number, and a buffer to receive the data. A local copy of the header sector map is used to convert from logical to real sector number, and the data is transferred from the file server to the buffer.

If the file lock has been broken (i.e., the file server has released the lock for some reason such as client inactivity), all files opened for this transaction are unlocked, the state changes to **COMPLETED**, and an error code signifying *Transaction Aborted* is returned. The transaction system makes no assumptions about how a client program will deal with such aborts. In typical usage, they will probably be rare and the client can simply treat them like a software error requiring a program restart.

WRITE

WRITE receives as arguments a file handle, a logical sector number, and a buffer containing the data to be written. It first uses the local copy of the file free-list to allocate an unused real sector. Then, the local copy of the file sector map is checked to see if the logical sector being written already exists. If so, its real sector number is recorded in a separate list to be merged with the header free-list during the completion of the transaction. Finally, the local copy of the sector map is updated to indicate the new mapping, and the buffer is transferred to the file server. As in **READ**, the transaction is aborted if the file lock has been broken.

Notice that all of the useful information from before the transaction is left untouched. Only local copies of headers are changed, and writes go to unused real sectors. The addition of sector numbers to the header free-list is delayed until **END TRANSACTION** so that subsequent writes during this transaction will not use them.

Figure 3-2: Paxton's Transaction Primitives

ABORT TRANSACTION

All files opened for this transaction are unlocked and the state of the transaction is changed to **COMPLETED**.

END TRANSACTION

There are three cases to consider for **END TRANSACTION** depending on whether zero, one, or more than one file was written. In all cases, the transaction is immediately aborted if a broken lock is discovered before the state becomes **COMMITTED**.

If no files were changed, **END TRANSACTION** simply unlocks all the files that were opened, changes the state to **COMPLETED**, and returns.

If a single file was modified, then after the read-only files are successfully unlocked (i.e., their locks were not found to be broken), the modified file has its header written and its lock released. In both this case and the previous one, the intentions file is not required and no accesses are made to it.

If multiple files were written, **END TRANSACTION** must leave around enough information so that if it crashes after committing to make the changes someone else will be able to complete them. The intentions file is the place for this information. The sequence of steps is as follows:

1. Unlock all of the read-only files.
2. Lock the intentions file. (This is the first action involving the intentions file for this transaction; up to this point all the activity has been with the data files.)
3. Mark each modified file as being changed in this transaction by placing in its header the new transaction number and the identifier for the intentions file.
4. Write copies of the new headers to the intentions file. The header free-lists are first updated to include the real sectors which are now unused.
5. Write the list of changes to the intentions file with the transaction number updated and the state variable set to **COMMITTED**. There is now a commitment to completing the transaction rather than aborting it. If a broken lock is discovered after this point, crash recovery will be invoked to finish the transaction.
6. Write the new headers to the data files saying that they are not involved in any transaction. Unlock each file after writing its header.
7. Write the intentions file header with the state variable set to **COMPLETED**, and then unlock it.

Note that the state recorded in the intentions file is either **COMMITTED** or **COMPLETED**. The **STARTED** state is not recorded.

Figure 3-3: Paxton's Transaction Primitives, continued

the primary server or coordinator and is responsible for coordinating the transaction; the other servers are designated as workers. The coordinator maintains a list called a worker list that describes the actions that must be performed at the various servers. Each worker also has an intentions list that describes actions that must be carried out at that site.

Each intentions list is designed so that partially carrying out the list several times and then finally completing it is the same as performing it exactly once. Intentions lists are written and erased as atomic acts. A suggested implementation is to represent files as blocks of pointers to data pages and an intentions list as a list of new pointer values. A transaction is then implemented by writing data only in newly allocated disk pages and collecting the addresses of the new pages in an intentions list, the new addresses can then be merged into the file pointer block at the time of the close transaction request.

The integrity of intentions lists and file pointer blocks is further enhanced by the use of stable storage, which is basically a technique for maintaining two copies of critical disk pages in such a way that only one copy is being updated at any given time. This aids in system recovery if a crash should occur when one of these blocks is being updated.

DFS effectively makes a file system spread over multiple servers usable as if it were a single file system.

DFS dialogues are of the form request message (from client to server) followed by either result message or reject message (from server to client). A typical request message might look like:

read data (file UID, index of first byte, number of bytes).

The UID is a unique identifier of the file (that is unique across all servers); the index of the first byte is essentially a page number plus offset.

The typical response to a read data message would be a here's data message of the form:

here's data(bytestream)

The integrity of the actual data transfer is the responsibility of the

underlying communication system.

DFS thus

- o implements files as simple sequences of bytes or pages, and
- o provides some simple file locating facilities that can be used to find the server holding a file given the file's unique identifier (UID).

The client (application) programs:

- o handle data structuring beyond the linear sequence of bytes level,
- o keep track of where files are located, and
- o may cache some data locally to provide faster access.

DFS is essentially the set of file servers and in a cooperative manner:

- o provides reliable file storage,
- o implements the "atomic property",
- o resolves concurrent access requests,
- o deals with deadlocks (i.e. locks held too long), and
- o decides which requests to honor (access control).

The set of client machine front-ends:

- o handle functions that require knowledge of application domains,
- o provide interpretation of the format and logical structure of files, and
- o provide the end user interface.

In this sense the Xerox systems put most database management functions in the client machines and use the servers essentially as back-end storage managers.

January 1980

4 Tandem GUARDIAN/EXPAND/ENSCRIBE

While not currently designed explicitly for a distributed data base system, the Tandem architecture provides an unusually resilient framework for data storing through a combination of operating system, network, and data base/file system facilities and a highly modular fault tolerant architecture.

The Tandem computer is basically a tightly coupled multiprocessor system consisting of up to 16 processor-memory pairs connected to each other by a high speed pair of buses, collectively called the Dynabus. The processors are connected to disk and other device controllers via multiple I/O buses. Processors, device controllers, and disks can all be dual ported and all can be duplicated to provide enough redundancy so that the system can continue to function if any single component fails. (Some of the redundancy is made optional for cost reasons.) Because of this redundancy, Tandem Computers refers to this as their NonStop™ System Architecture.

A typical configuration is shown in Figure 4-1. In the figure three Tandem modules are connected to four dual ported device controllers by three I/O buses and to each other by the dual path DYNABUS. Each Tandem module consists of a DYNABUS control interface, a CPU, up to two megabytes of local memory, and an interface to an I/O channel. Disks, printers, and/or other peripheral devices can be attached to the device controllers.

Configurations can consist of between 2 and 16 processors each with roughly the capacity of a PDP-11/70, but with the ability to support up to 2 Megabytes of memory, a 26 Mbyte/sec Dynabus consisting of two 13 Mbyte/sec ribbon cables integral with the computer chasis, and a number of 4 Mbyte/sec I/O channels. The processors are usually adjacent and cannot be more than a few feet apart, but several of these clusters can be linked together using the EXPAND networking facility.

The redundancy described above can be extended to include duplication of data through the use of mirrored discs. This is illustrated in Figure

4-2. In the figure separate copies of critical files are kept on two discs that are identical images of each other. Two paths are provided through separate disc controllers to get to each disc. When this option is chosen, duplicate copies of entire discs are kept online and all updates are made to both copies. This updating is done by system software transparent to the user. If a disc is lost because of a head crash or similar problem, a second copy is still available to allow normal operations to continue.

The GUARDIAN operating system functions as a traffic controller and handles communication between processors and between processes.

This entails:

- o relaying messages to the correct processor and verifying that they are received correctly.
- o directing the messages to the appropriate program or device within the receiving processor, and
- o detecting and responding to errors or nonresponsiveness anywhere within the system. This includes logging failures and rerouting communications away from faulty modules.

A summary of the main operating system primitives and functional modules is shown in Figure 4-3.

The NonStopTM option of the operating system supports the existence of backup versions of application programs running on a second processor. Modified portions of memory are copied between processors at checkpoint intervals; these occur just before a write to a peripheral device, just after a read and at user specified times. If the operating system detects that the primary processor has failed, the program is automatically restarted on the backup processor from the last checkpoint. Until this happens the backup processor can be used to run other programs with only limited overhead (due to memory transfers) from the backed up application program.

The ENSCRIBE data base/file management system is a complex access method handler with a plethora of features including:

- o key-sequenced and relative file structures

- o multi-key ISAM with up to 255 secondary indexes
- o approximate and generic key matching
- o cache buffering of physical data blocks to main memory in a way that is transparent to the user
- o data and index compression and
- o record and file locking

Effectively one copy of the ENSCRIBE facility runs in a Tandem-16 cluster. If several clusters are linked together using the EXPAND facility or if several application programs are accessing or updating the same data, it is the responsibility of the applications to ensure consistent use of the data base. ENSCRIBE handles locking on a per record or per file basis. Locking on a transaction basis must be provided by individual applications that need them.

If a data base is spread over several clusters, ENSCRIBE treats it as if it were fully partitioned. Any consistency controls on redundant copies of data (except for the redundancy provided by mirror disc duplication) must be provided by user programming.

A typical single cluster ENSCRIBE configuration is shown in Figure 4-4.

In summary, the major software components of Tandem's environment are:

- o GUARDIAN, the operating system,
- o ENSCRIBE, the file/data management system, and
- o EXPAND, the networking system.

These products are all in current commercial use. While the announced commercial products do not provide the full functionality of a distributed DBMS, they provide a foundation upon which the remaining components could be built.

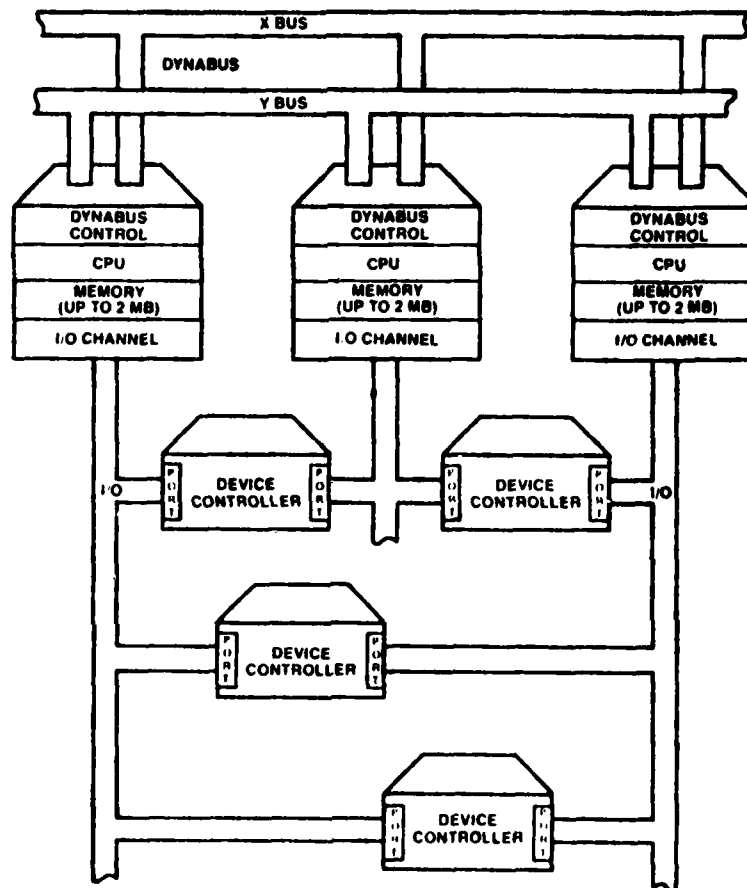


Figure 4-1: The Tandem NonStop™ System Architecture

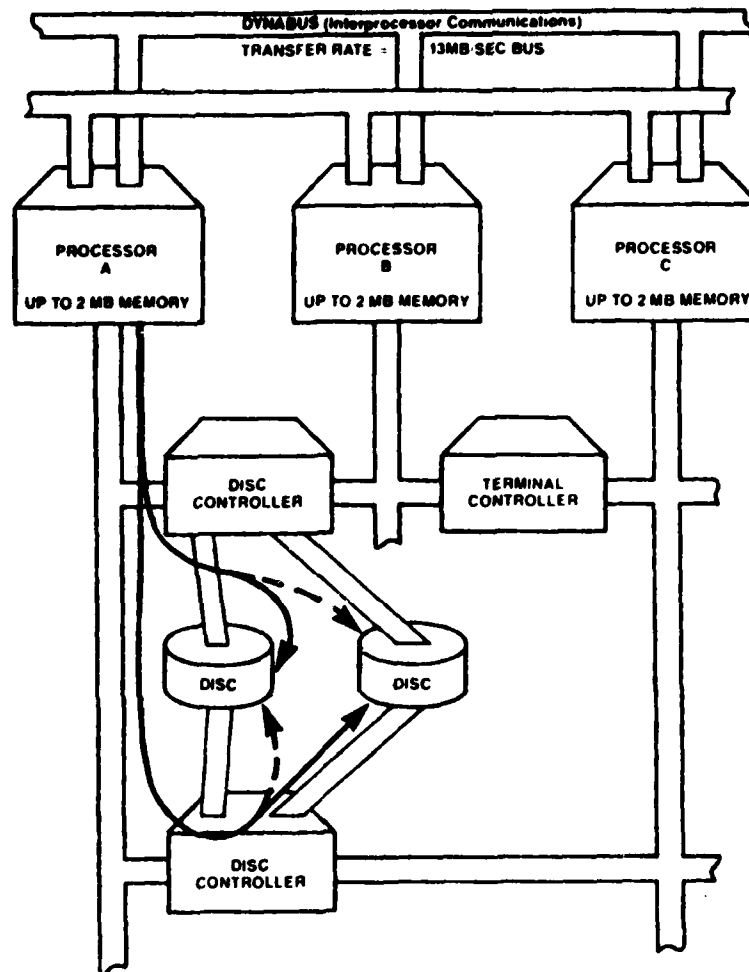


Figure 4-2: Mirrored Disc Volumes on a Tandem System

Guardian Operating System

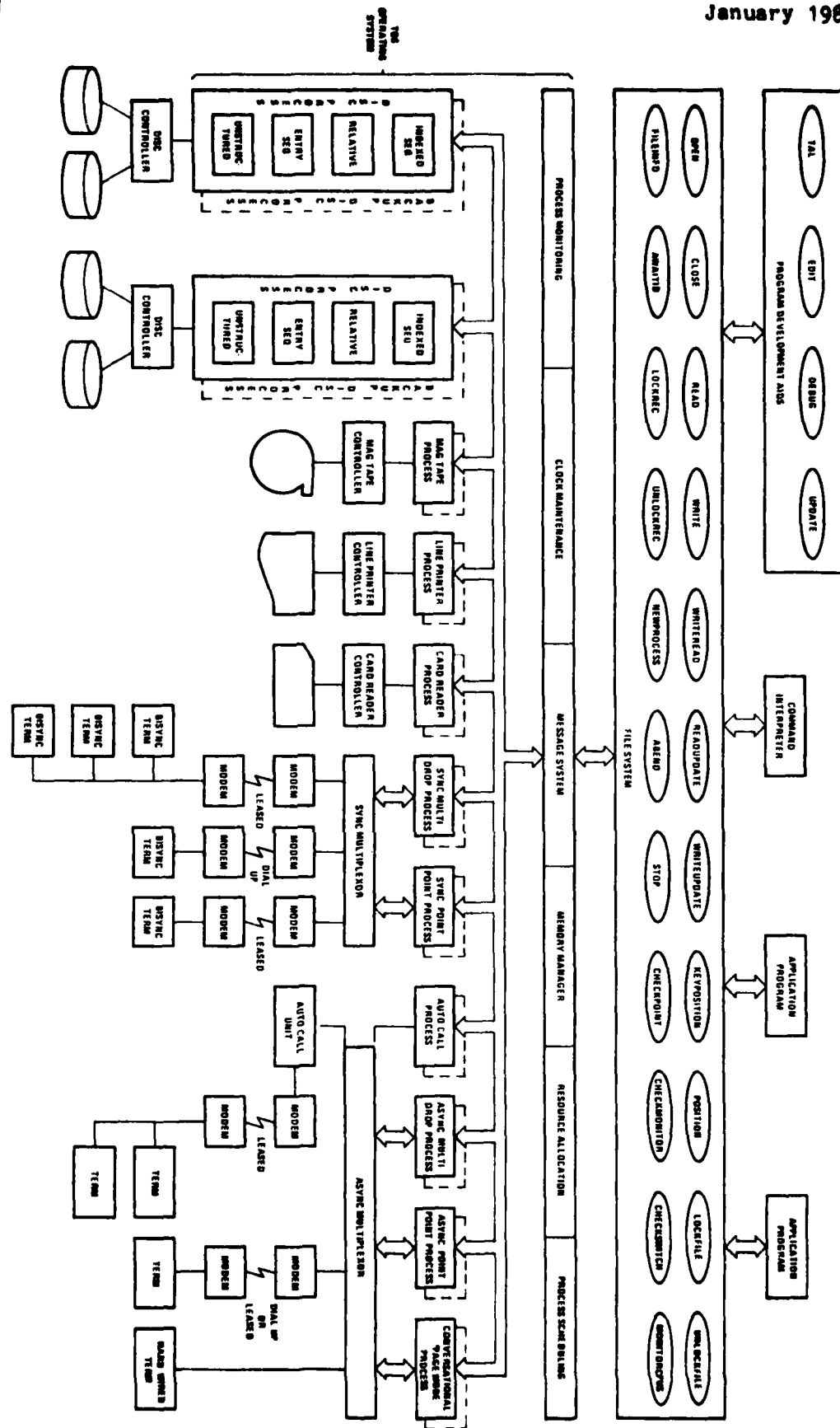


Figure 4-3: Logical Organization of the GUARDIAN Operating System

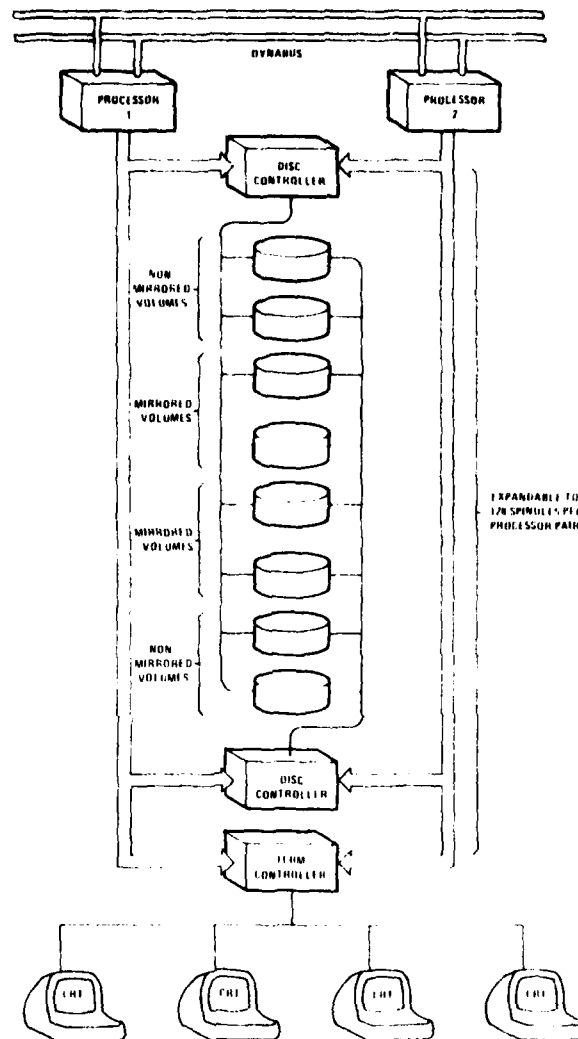


Figure 4-4: A Typical ENSCRIBE file configuration

January 1980

5 SDD-1

SDD-1 [Rothnie 77], [Rothnie 79], [Rothnie 80] is a distributed database management system developed by the Computer Corporation of America (CCA) of Cambridge, Massachusetts and is accessible over the ARPANET. Its component pieces can be dispersed geographically and may contain redundant data. In order to handle this redundancy in a manner transparent to users, CCA introduced a number of innovations into existing DBMS technology. The most important of these innovations and related design decisions are:

- o the separation of distributed data base management into local and global parts. This makes it at least theoretically possible to develop these two pieces separately and to substitute algorithms at the local level without affecting the global data management strategies. SDD-1 extends this breakdown further by dividing global data management into a piece called a Transaction Module concerned with transaction processing and a piece called RELNET (for Reliable Network) concerned with certain reliability enhancements to the communication facilities of the network that insure the atomicness of transactions (i.e. that either all the updates associated with a transaction are executed or none are).
- o a three phase division (Read/Execute/Write) approach to processing database transactions. This allows the issues of concurrency control, distributed query processing and reliable posting of updates to be handled in separate phases of a database transaction and simplifies the process of maintaining database consistency.
- o the use of conflict graph analysis as an alternative to global database locking. By dividing database transactions into classes and preanalyzing the classes for potential conflicts, SDD-1 can substitute a combination of protocols and local locking for the more usual global locking techniques. The forced delay of certain transactions inherent in some of the suggested protocols has an effect essentially equivalent to global locking.
- o the handling of directory information as ordinary user data. This allows the amount of information stored redundantly at different sites to be tuned to suit particular applications. In addition, directory locator information (a directory to the directory) is stored at each data module.
- o the use of three mechanisms to ensure reliable writing of transactions:

- o spooled files to guarantee that updates will be delivered in the proper order to sites that are temporarily down or disconnected from the network.
- o a variation of two-phase commit to provide a convenient way of backing out of a transaction that cannot be completed because of failure of the node processing a transaction.
- o unique timestamps to ensure that the effect of transactions will be the same as if they were processed in serial order.

5.1 Overall Architecture

The overall architecture of SDD-1 is shown in Figure 5-1. SDD-1 is composed of semi-independent components called datamodules linked together via a communications network. Each of the data modules is equipotent (equally powerful), in the sense that there is no central point of control or vulnerability.

Each datamodule is composed of two parts:

- o the global data manager (GDM) and
- o the local data manager (LDM).

The GDM is responsible for

- o parsing user requests into a canonical internal form,
- o determining which data modules contain data involved in the requests,
- o in general, planning and controlling the distributed execution of the transaction and ensuring that data transfers take place in a consistent manner.

The local data manager is responsible for managing data stored at its node (e.g. retrieving and modifying locally stored data) and to a large extent can be unaware of data distribution issues. The initial implementation uses the CCA Datacomputer [Datacomputer 78] as its local data manager. This implementation supports a relational data model and features a high level query and data manipulation language, Datalanguage, which is similar

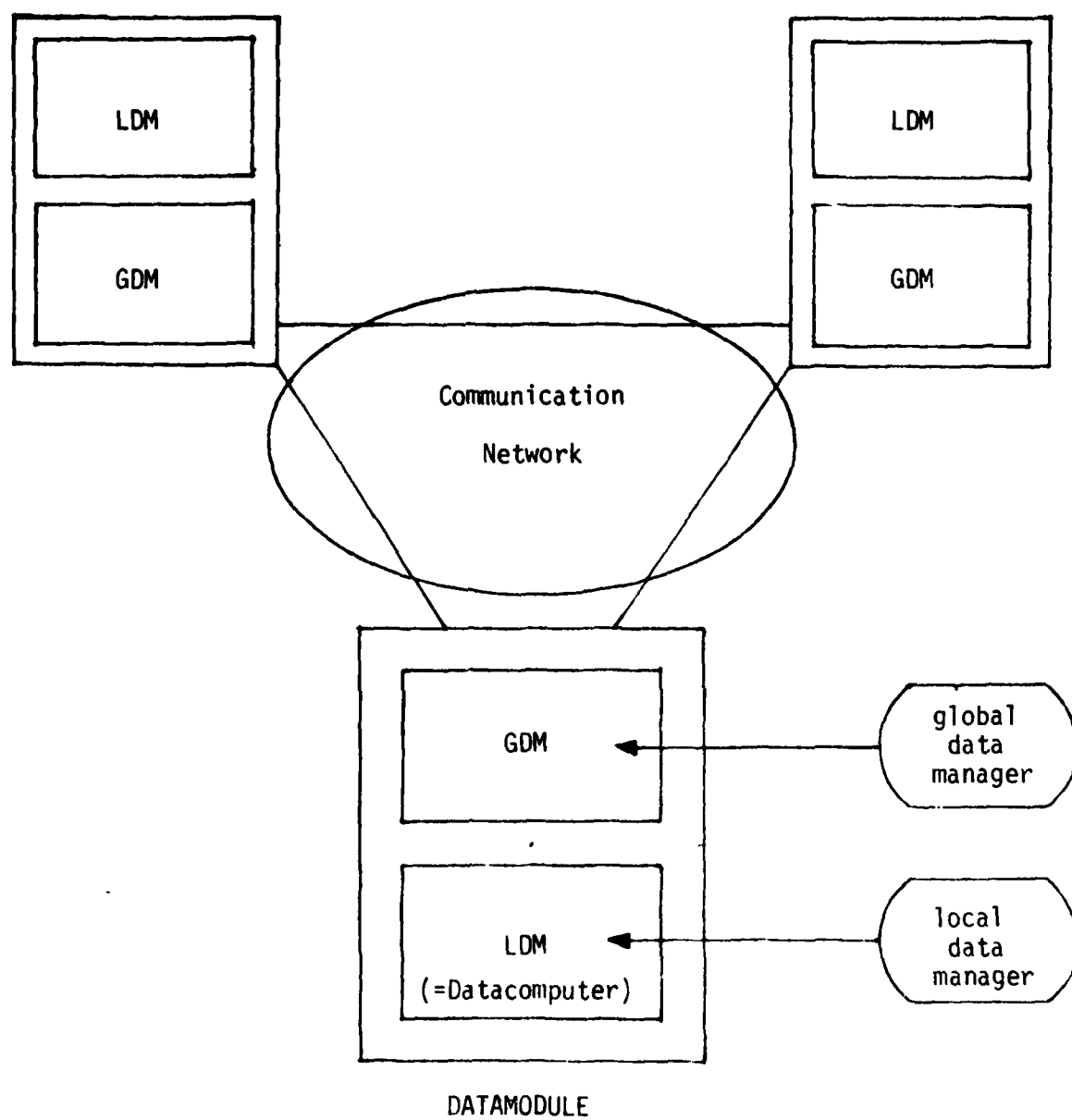


Figure 5-1: Decomposition of SDD-1 into Local and Global Parts

to QUEL [Held 75].

5.2 Logical Structure

The SDD-1 Architecture divides distributed data base management into three functional areas that are treated as separate subsystems. (See Figure 5-2.) These subsystems are composed of three types of modules (circles in Figure 5-2):

- o Transaction Modules (TMs) plan and control the distributed execution of transactions.
- o Local Data Modules (DMs) are effectively backend data base management systems that manage local data bases in response to commands from transaction modules.
- o A Reliable Network (RelNet) provides enhancements to the underlying communications network to connect the TMs and DMs in a robust fashion, providing backup mechanisms to compensate for intermittent failure of system components.

Transaction Modules (TMs) perform four functions related to the planning and control of the distributed execution of transactions:

- o The Fragmentation Function translates queries on relations into queries on logical fragments and decides which instances of stored (logical) fragments to access.
- o Concurrency Control synchronizes a transaction with all other active transactions in the system.
- o Access Planning compiles a transaction into a parallel program that can be executed cooperatively by several DMs.
- o Distributed Query Execution coordinates execution of the compiled access plan, exploiting parallelism whenever possible.

Data Modules (DMs) respond to four types of command from the transaction modules to support transaction processing:

- o Read - part of the DMs database into a local workspace at the DM.
- o Move - part of a local workspace from this to another DM.

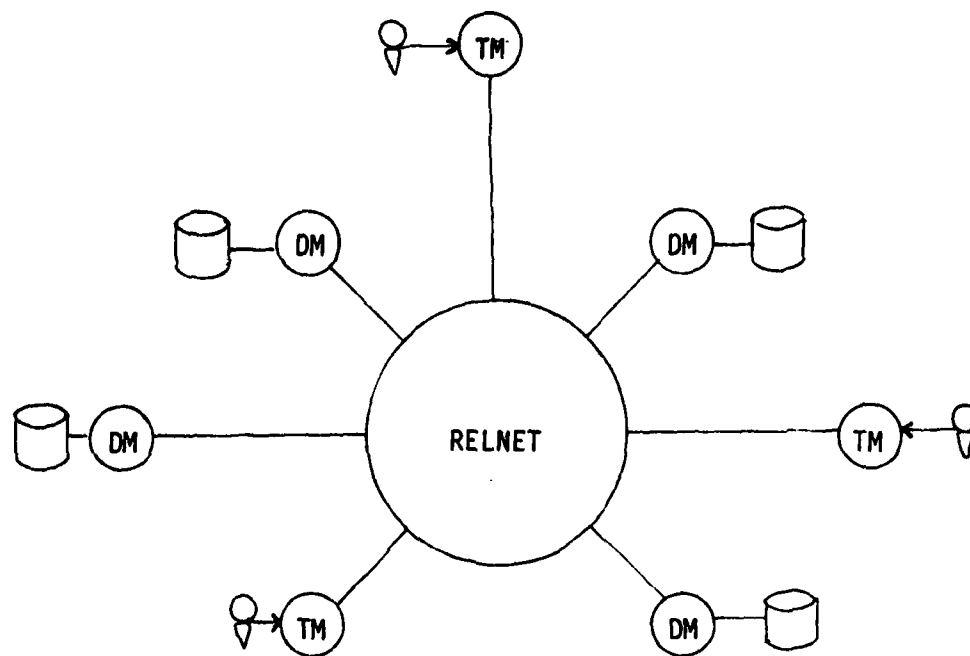


Figure 5-2: SDD-1 Logical Architecture

- o Manipulate - data in a local workspace at the DM.
- o Write - part of the local workspace into the permanent database stored at the DM.

The Reliable Network (Relnet) provides four services that support distributed DBMS reliability:

- o Guaranteed delivery allows messages to be delivered even if the recipient is down at the time the message is sent and even if the sender and receiver are never up simultaneously.
- o Transaction control posts updates at multiple DMs in such a manner that either all DMs post the update or none do.
- o Site monitoring keeps track of which sites have failed and informs other sites that might be impacted by the failures.
- o Network clock is a virtual clock kept approximately synchronized at all sites.

5.3 Run Time Scenario

SDD-1 transactions are handled as a three phase process involving:

- o Reading data from the database. The data gathered in this way is called the transactions read-set.
- o Executing the transactions by manipulating data from the read-set obtained in phase 1, and
- o Writing the modified data back to the database or to output devices as an atomic operation. This modified data is called the transaction's write-set.

The read phase is mainly concerned with concurrency control. In it, the transaction Module (TM) supervising the transaction determines which data from the database needs to be accessed (i.e. the transaction's read-set) and decides which stored fragments to access to obtain that data (if the data is stored redundantly). Then the TM issues read commands to the Data Modules that control these fragments. This causes those DMs to set aside private copies of the fragments to use during the subsequent processing phase. No data is actually transferred between sites during the

read phase.

The second or execute phase handles the distributed query processing. In this phase, the Transaction Module compiles the transaction into a distributed program that takes the distributed workspace created by the Read phase as input. The compiled program consists of move and manipulate commands which cause the Data Modules to perform the transaction in a distributed fashion. The TM supervises the compiled program to ensure that commands are sent to DMs in the correct order and also handles any run time errors. The output of this stage is a list of data items to be written into the database (updates) and/or displayed to the user (retrievals). This output list is produced in a workspace (temporary file) at one DM and is not yet written into the permanent database.

The final or write phase is responsible for copying modified data into the permanent database in a consistent (i.e. atomic) manner and displaying retrieved data to the user. For each data item to be modified (e.g. for each entry in the output list), the TM determines which DMs contain copies of that data and directs the DM controlling the output list to send the updated data to each of these DMs. It then issues Write commands to each of these DMs thereby causing the new values to be written into the permanent database.

5.4 Methods and Algorithms

5.4.1 The concurrent update problem

SDD-1 uses serializability as its criterion for database correctness in the presence of a sequence of overlapping transactions. This criterion requires that the concurrent execution of a set of transactions be equivalent to executing them one at a time, serially, in some order. Thus, if each individual transaction leaves the database in a consistent state then a concurrent serializable sequence of transactions will as well.

The usual technique for ensuring serializability is database locking (with some granularity). SDD-1 avoids this through the use of a mechanism called conflict graph analysis. This technique involves analyzing classes of transactions offline to determine what level of synchronization is

needed between transactions of different classes. A graphical technique based on analyzing conflicts between data written and data accessed by transactions is used for doing this. The details of this technique are described in [Bernstein 78].

Protocols based on timestamps are used at execution time to prevent conflicts in cases where they are liable to occur.

These timestamps are made unique by concatenating a TM identifier to the right (low order bits) of the network clock time and limiting each TM to generating at most one transaction per clock tick. The appropriate (transaction) timestamp is attached to all Read and Write commands sent to DMs. In addition, each Read command contains a list of transaction classes that conflict "dangerously" with it. When a DM receives the Read command, it defers the command until it has processed all earlier Write commands (those with smaller timestamps) and no later Write commands (those with larger ones) from the TMs for the specified classes. The above technique is made to work because

- o each TM sends its Write commands to DMs in timestamp order,
- o the Reliable Network guarantees that messages are received in the order sent,
- o idle TMs periodically send null (empty) timestamped Write commands, and
- o DMs can explicitly request a null Write from a TM that is slow in sending them.

Four protocols (with different levels of synchronization) are provided in the initial version of SDD-1 along with a protocol table for choosing between them.

5.4.2 Distributed Query Processing

SDD-1 uses an access planning technique which attempts to minimize communication costs as its primary optimization goal with local processing cost minimization a secondary goal [Wong 77]. This makes it particularly suitable for geographically distributed networks where communication cost

are high. In addition, its access planner seems to assume use of a relational data model. It seems likely that different access planning strategies would be chosen for a local area network, but the overall architecture and many of the design decisions seem applicable to both types of network.

The programs produced by the Access Planner can be represented as data flow graphs [KARP 66]. To execute the program, the TM controlling a transaction issues commands to the DMs involved in each operation as soon as all predecessors of the operation are ready to produce output. The effect of the execution is to create at the final DM a temporary file to be written to the database or displayed to a user.

5.4.3 Reliable Writing

The goal is to ensure that failures of system components cannot cause some DMs to install updates while others do not. Three types of failure can occur:

- o failure of a receiving DM
- o failure of a sender
- o failure of the communications medium causing partitioning of the network.

SDD-1 protects against the first two by guaranteed delivery and transaction control respectively; problems of network partitioning are handled by manual intervention.

SDD-1's guaranteed delivery mechanism supplements normal network message integrity mechanisms by handling the case of sender and receiver not being up simultaneously to exchange messages. To do this, Relnet uses processes called spoolers that have access to secondary storage and manage FIFO message queues for failed sites. Any message destined to a failed DM is delivered to its spooler instead. Protection against spooler failures is provided by employing multiple spoolers. When a failed DM recovers, it accesses the appropriate spooled queue to bring its database up to date.

Transaction control protects against a TM or final DM failing during the write phase after transmitting some of its updates but not all of them. It does this by effectively making the write phase an indivisible (atomic) operation using a variant of "two-phase commit" [Gray 78]. During phase 1, the final DM transmits all update data but the receiving DMs do not install it in the permanent database yet. During phase 2, the final DM sends commit messages to all DMs involved in the transaction. Once a commit has been received by at least one DM, the transaction will be completed even if the Final DM fails; if the Final DM fails before sending any commits, the whole transaction will be aborted. A consultation protocol is used to resolve which case has occurred if DMs have not received commits after a reasonable interval.

As an alternative to processing Write commands in timestamp order, SDD-1 uses a novel technique first proposed by R.H. Thomas of BBN [Thomas 79] for using timestamps attached to each data item in the data base to improve update performance. Each physical data item in the database is timestamped with the time of the most recent transaction that updated it. When an update is received at a DM as part of a Write command, the value in the database is modified if and only if the timestamp stored in the database is less than the timestamp of the Write command. This rule ensures that recent updates are never overwritten by older ones and that the net effect is the same as processing Write commands in timestamp order³

³It, however, depends on the strict division of operations into a read/modify/write sequence, as occurs in SDD-1, and could not be used if modify type atomic primitives, such as "increment data item by 1" were allowed.

6 Distributed INGRES

INGRES [Held 75], [Stonebraker 76] is a relational data base management system that runs under the UNIX operating system on DEC PDP-11 and VAX hardware. Distributed INGRES [Stonebraker 77], [Stonebraker 78] is a special version of the INGRES data base management package that contains extensions to support the linking of copies of INGRES running at different sites and the division of INGRES into master (front-end and control) and slave (back-end data manager) pieces. The slave INGRES can run on the same or a different computer from the master INGRES. The slaves are initiated (spawned) by the masters at system configuration, system generation, or user logon. The logical architecture of Distributed INGRES is shown in Figure 6-1. In the figure one master INGRES controls and is supported by three slave INGRESes.

The master INGRES serving a user

- o handles the user interface to the distributed data base system,
- o spawns slave INGRESs at sites where data base fragments exist and handles synchronization between them during processing of data base queries and updates,
- o processes user queries by sending commands to the slaves which then access or update the local data.
- o handles update concurrency control, crash recovery and other integrity constraints.

A slave INGRES needs to be initiated at a particular site only after the first query that needs data at that site is initiated. It need be initiated only once (at system generation or user logon time). A slave INGRES

- o handles all accesses and updates to the (local) piece of the data base it controls
- o responds to commands from master INGRESs at its own or other nodes.

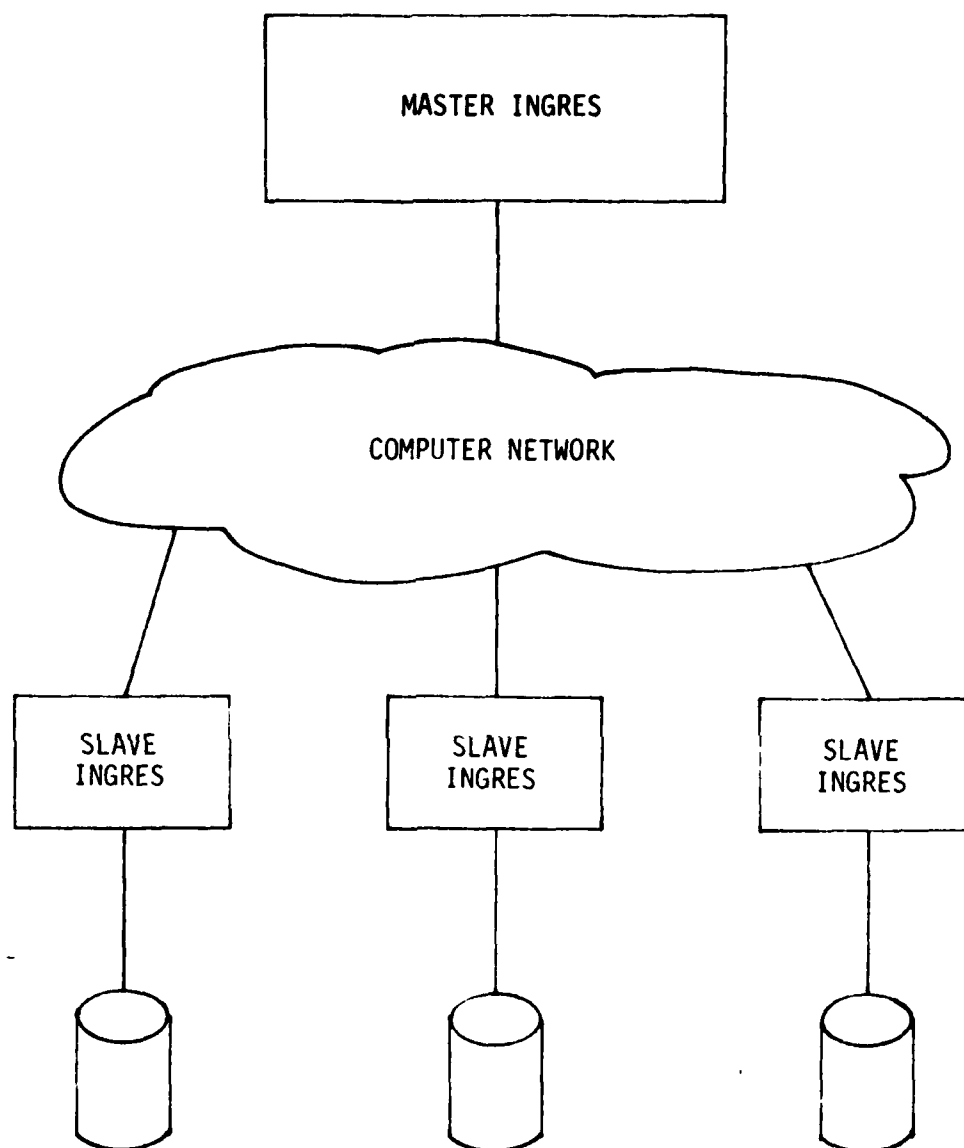


Figure 6-1: Logical Organization of Distributed INGRES

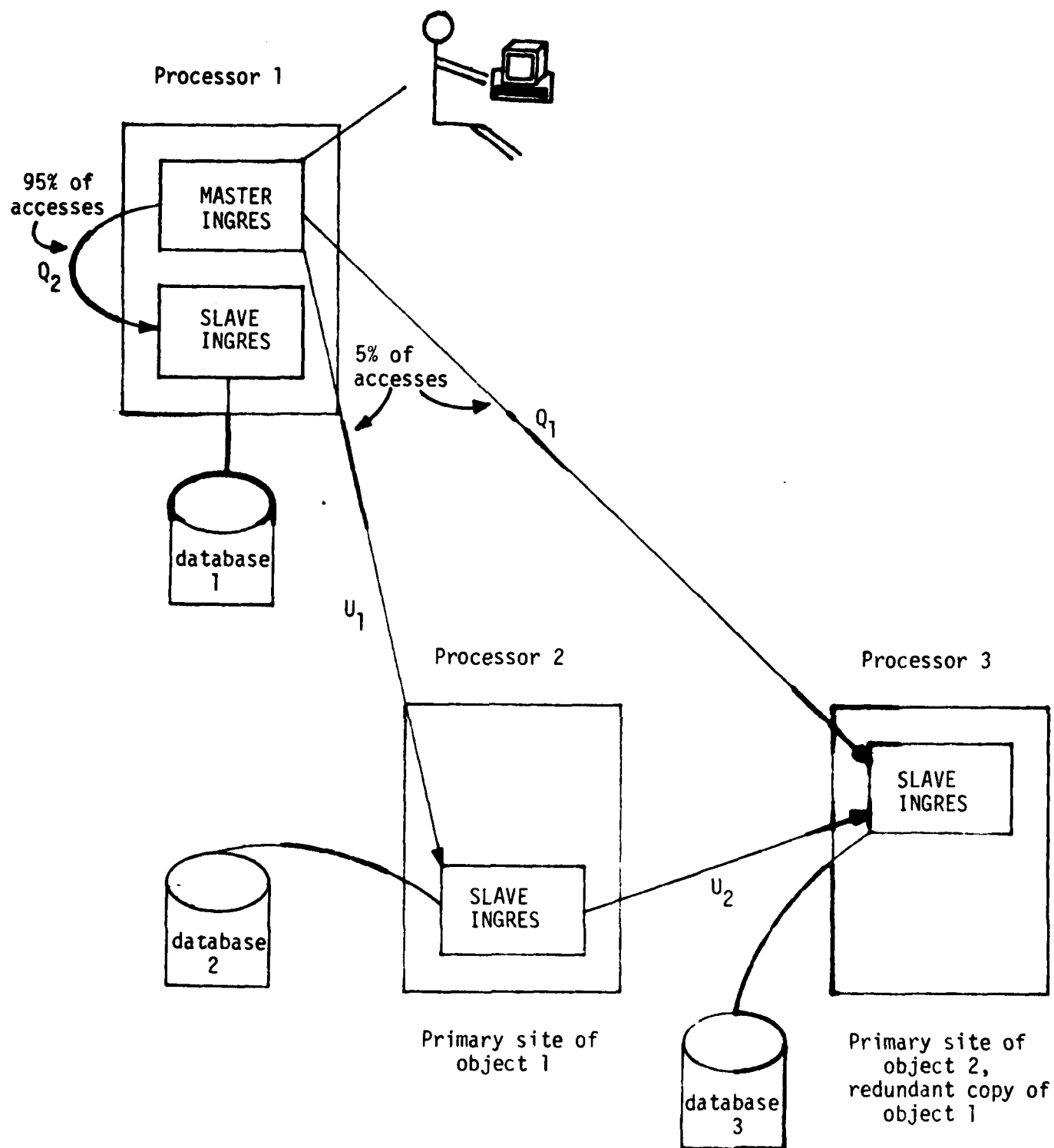
- o sends results of updates to other slave INGRESs which have duplicate copies of the same data.

Distributed INGRES uses the primary site model [Alsberg 76] for distributed control of data. In this model, each object has a primary location to which all updates for that object are directed. Other copies of the object are then updated to make them consistent with the primary site. Different objects may have different primary sites. This provides a concurrency control mechanism for redundant copies of the same data but does not address the problem of ensuring that a transaction that updates multiple objects is treated as atomic (indivisible).

An object in INGRES is a subset of the rows of a relation, what SDD-1 calls a logical fragment. Each of these pieces of a relation can have a different primary site and/or different number of redundant copies.

When a primary site slave INGRES receives an update request, it updates its copy of the object and then forwards the update to all other copies of the object. Figure 6-2 shows the data access process in a typical configuration. In the figure an end user logged onto processor 1 sends query requests to a master INGRES on his local processor. The master INGRES parses his requests and directs them to the appropriate slave INGRES. In the case of queries, the slave INGRES obtains the data from its associated data base and returns it to the master INGRES. In the case of updates, the request is forwarded to the slave INGRES at the primary site for that data. The primary site slave updates its data base and then forwards copies of the modified data to all other slave INGRESes maintaining copies of the same data. A variation of two-phase commit is used to ensure consistency of redundant copies (see Chapter 5 for a discussion of two phase commit). Distributed INGRES was optimized to work effectively when approximately 95% of accesses are local and only 5% are to data at remote nodes.

Deadlock detection and resolution is handled primarily by local concurrency controllers that run at each site. In addition, there is a daemon (asynchronous process) called SNOOP, which maintains a table of lock requests for the entire network. When a local concurrency controller detects that a request at its site can not be satisfied, it sends a message



MASTER INGRES = front end

SLAVE INGRES = back end

Figure 6-2: Distributed INGRES Logical Interactions

to the SNOOP. The SNOOP adds the information to its wait for graph, which it then analyzes to detect potential deadlock conditions.

To protect against failure of a single node, Distributed INGRES spools messages destined to sites that are not accessible and has a recovery procedure which is executed when that node rejoins the network.

Distributed INGRES is designed for configurations where most transactions access data primarily at the node where they originate. Two sets of algorithms are provided. One set takes advantage of the local nature of most accesses to improve the performance of queries (at the risk of providing out of date or inconsistent information); the other set of algorithms provide better reliability at some performance cost. The details of these algorithms are discussed in [Stonebraker 78]; some of the tradeoffs in the initial design of Distributed INGRES are discussed in [Stonebraker 77]. Network support for Distributed INGRES is discussed in [Rowe 79]. Other approaches to distributing INGRES are discussed in the chapters on MUFFIN and DIRECT that follow.

January 1980

7 MUFFIN

MUFFIN (Multiple Fast or Faster Ingres) [Stonebraker 79] is a distributed data base machine-oriented version of INGRES designed to run on one or more local area networks.

A Muffin configuration is composed of a variable number of two types of processing nodes:

- o A-cells (application program cells) that run application programs and an Ingres front-end,
- o D-cells (data base nodes) that function as dedicated data base machines.

These two types of cells are connected into "pods" using a high speed bus or local network. The pods are (optionally) connected to other pods through gateways and lower speed communication links (see Figure 7-1). D-cells (the data base machines) consist of a bus interface tied to a conventional disk controller and secondary storage (disk or floppy disk) units (see Figure 7-2).

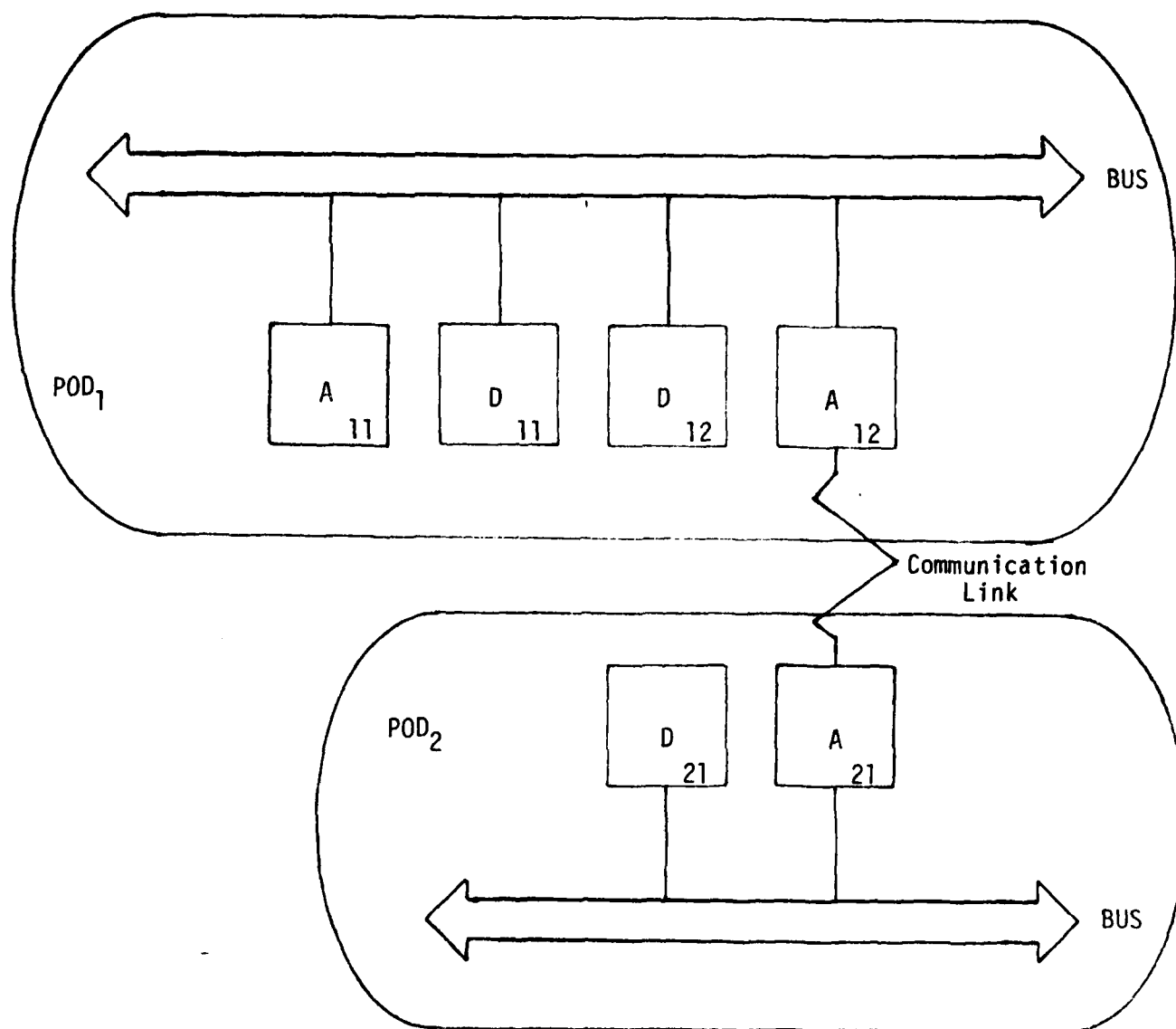
The D-cells are totally dedicated to running DBMS code. Their only interface to the outside world is through the bus; no terminals or other peripherals are directly attached to the D-cells. The D-cell is capable of accepting only data manipulation language (DML) commands written in QUEL [Held 75] and responding with data and /or status information. (Some Data Definition Language (DDL) interface is probably also needed, but is not described in the MUFFIN literature [Stonebraker 79].) Because of the specialized nature of the processing done and the absence of peripheral devices (other than those controlling the storage medium), operating systems support on the D-cells can be streamlined around the needs of the data base management system. Although other architectural optimizations of the D-cell seem feasible, none are really required or described in the Muffin paper.

A-cells, the application machines, are conceived to be either a smart

terminal, a personal computer or a general purpose processor running a time-sharing system. A typical A-cell configuration is shown in Figure 7-3. In the figure the user presumably a person at a terminal interfaces to an application program which in turn interfaces to an INGRES front-end, called Master INGRES. The Master INGRES ships data and DML commands to back-end INGRES data managers on D-cells and other INGRES front-ends (interfacing to other application programs) on other A-cells. These other INGRES modules are denoted as slave INGRESs in the figure.

The INGRES front-end (master INGRES) performs parsing, query modification for support of views, integrity constraints, predicate level storage protection, and utility functions. The slave INGRESs thus only need to support QUEL processing and some utility functions. To improve performance some portion of the data base may be coded on an A-cell. If this is done, the piece of INGRES running there would also have to support QUEL DML processing.

In the (experimental) implementation now underway (see Figure 7-4) the A-cells are VAX 11/780 processors running under UNIX, the D-cells PDP 11/70s, the local network a Farber type loop [Rowe 79] and the long haul network (connecting PODs) the ARPANET.



A = A Cell

D = D Cell

Figure 7-1: MUFFIN logical architecture

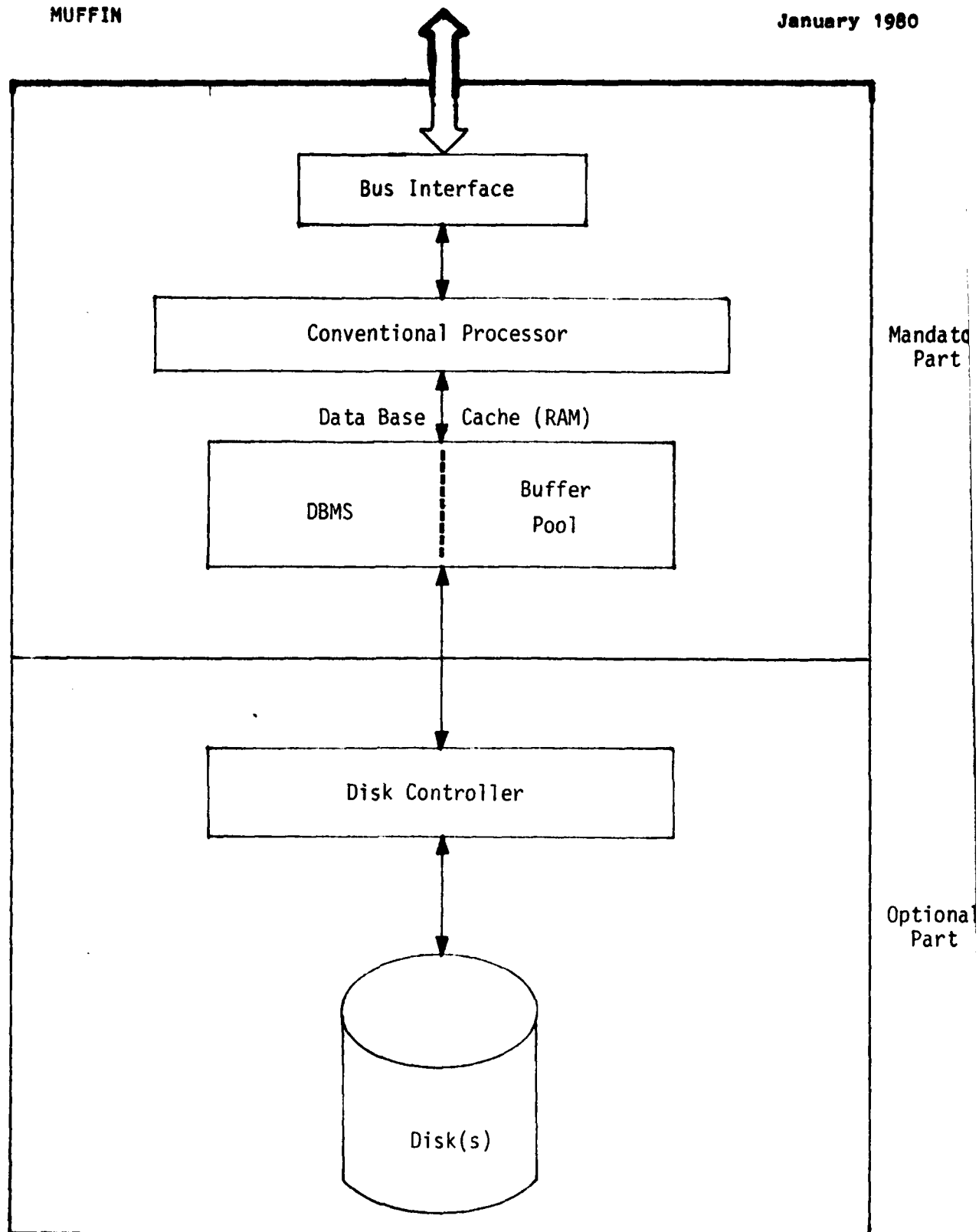


Figure 7-2: D-cell Physical Layout

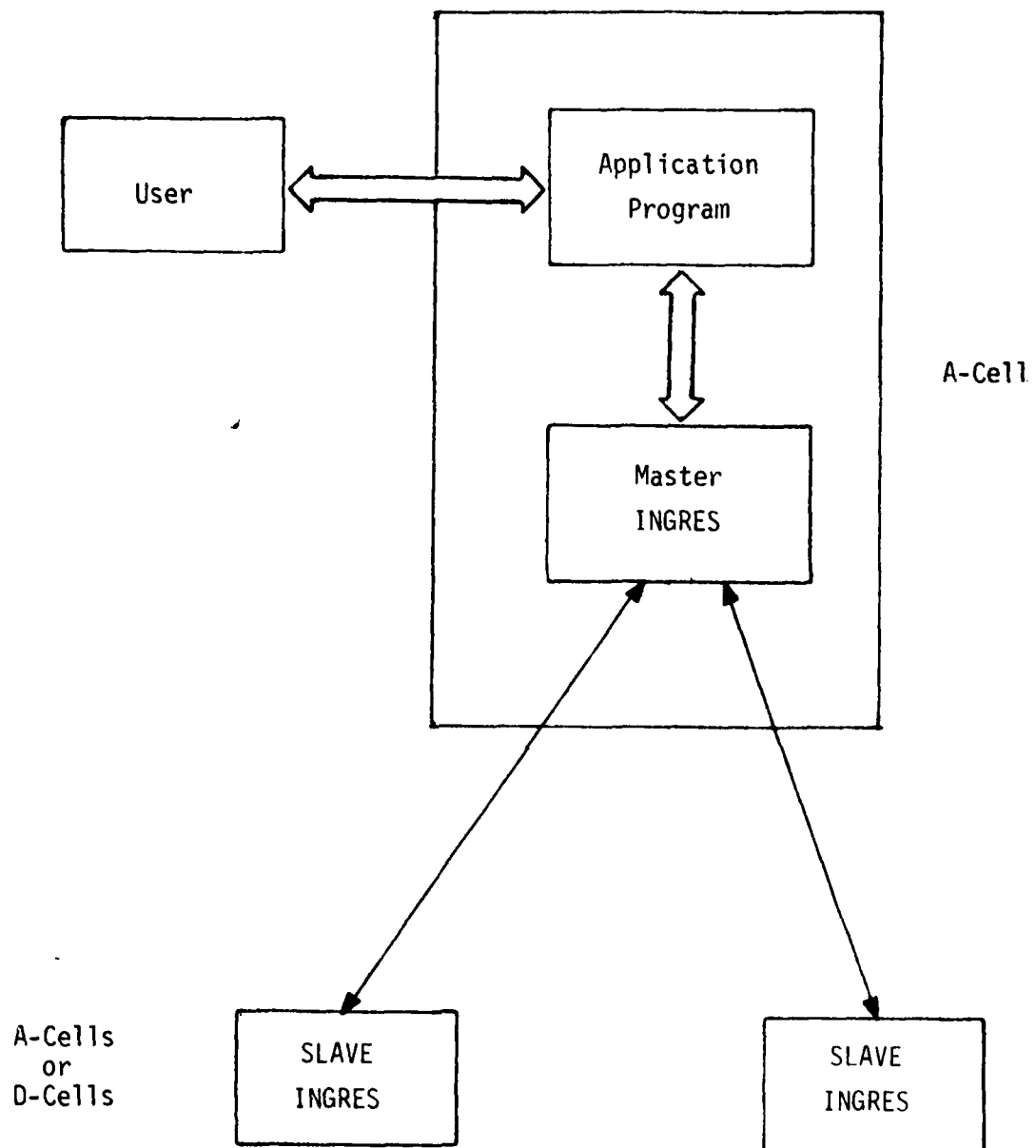


Figure 7-3: A-cell Run-Time Environment

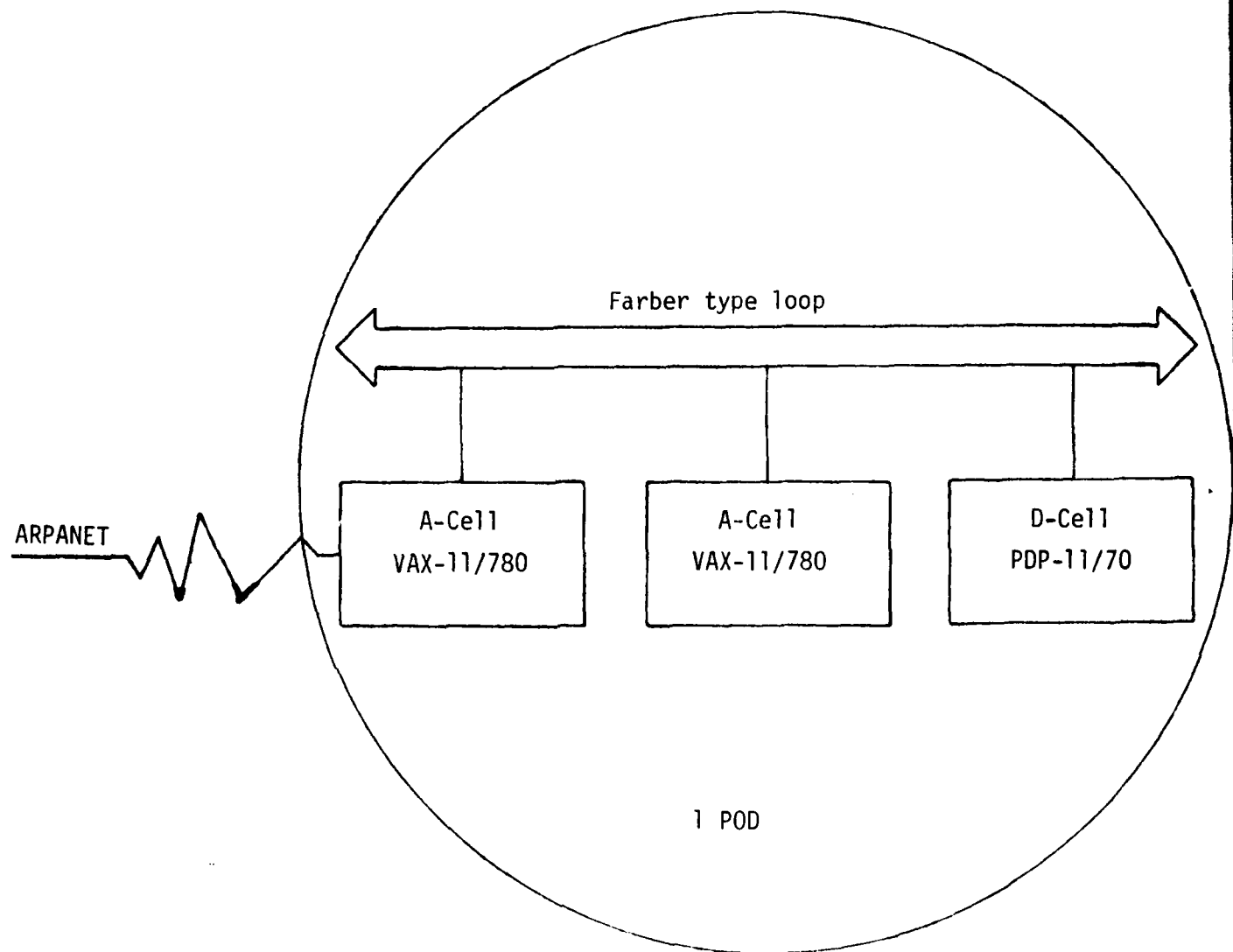


Figure 7-4: The MUFFIN Hardware

8 DIRECT

The DIRECT multiprocessor organization [DeWitt 79] developed at the University of Wisconsin by David DeWitt is basically a specialized MIMD (multiple instruction multiple data stream) architecture for the execution of queries and updates to a relational database. It consists of a host machine, a back-end controller (BEC), multiple microprocessors for executing query relations, mass storage, multiple CCD associative memories, and a cross-point switch interconnection matrix. (see Figure 8-1)

Users interact with the DBMS using an INGRES front-end running on a host processor (see Figure 8-1). The front-end is actually a modified version of INGRES that compiles user queries into sequences of relational algebra operations called "query packets". These query packets are actually unary/binary trees with relations at the leaves and relational algebra operators (JOIN, PROJECT, RESTRICT, UNION, INTERSECTION, DIFFERENCE, CROSS-PRODUCT, MAX, MIN, COUNT, etc.) at the non-leaf nodes. (See Figure 8-2).

After the query is compiled into the tree structured query packet by the front-end, the result is forwarded to the back-end controller over a DMA (DIRECT memory access) link. The back-end determines the number of query processors that should be assigned to execute the packet, manages the mapping of relations into associative memory, and distributes the query packet to the processors selected for execution. It can also run certain data base utilities for database creation/deletion and relation table management.

The query processors execute operations (JOIN, etc.) on the relations. Each relation is divided into pages so that multiple query processors can work in parallel performing the same operation on a given relation, work on different relations from the same query packet, or work on different query packets. Each query processor is thus performing one operation on a page from either one or two relations (depending on whether the operation is unary or binary) at any given time.

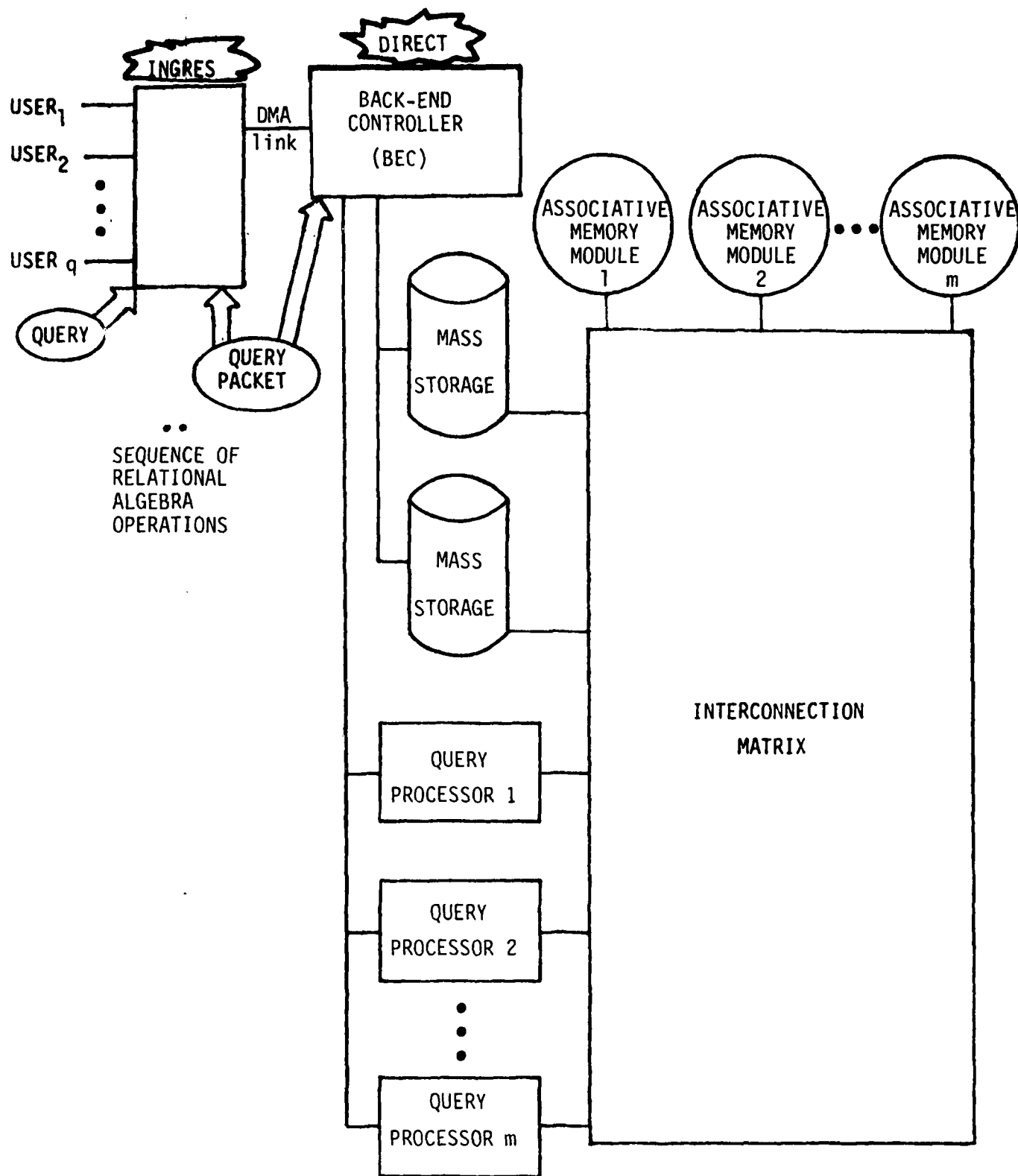


Figure 8-1: DIRECT system architecture

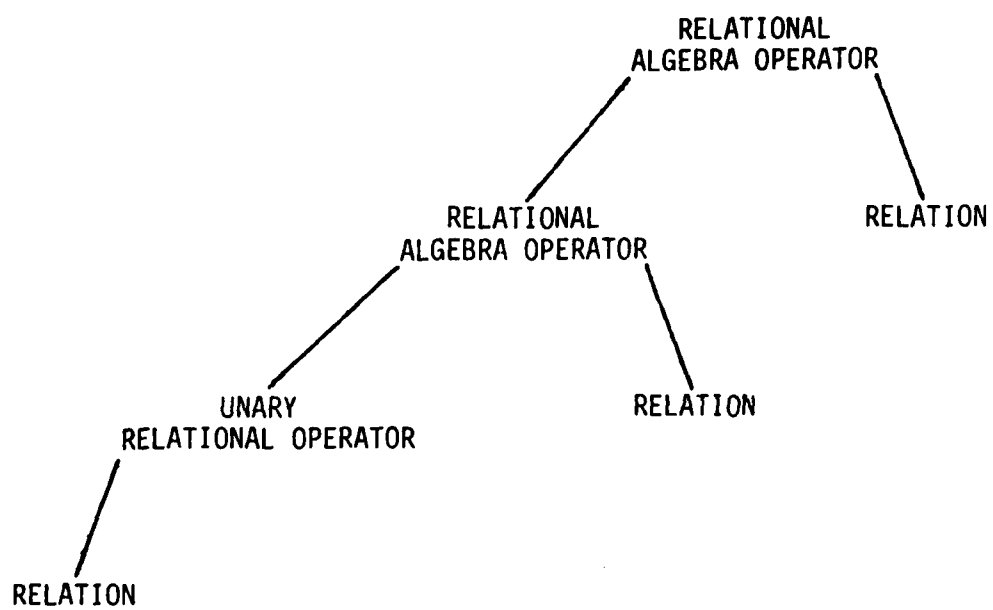


Figure 8-2: DIRECT tree structured query packet

The Back-End Controller (BEC) assigns processors to queries dynamically based on

- o the priority of the query,
- o the number and types of relational algebra operations it contains, and
- o the size of the relations referenced.

DIRECT treats an entire data base as one large linear virtual memory divided into 16K pages and maps these into associative CCD memories. The CCD memories function essentially as rotating registers that constantly "broadcast" their contents to any query processors that wish to "listen" by copying the CCD contents into their local memory. Since copying can begin at any place in a page, no delay is incurred by this approach.

Concurrent updates are handled by the BEC using locks on whole relations.

A diagram of the planned University of Wisconsin implementation is shown in Figure 8-3.

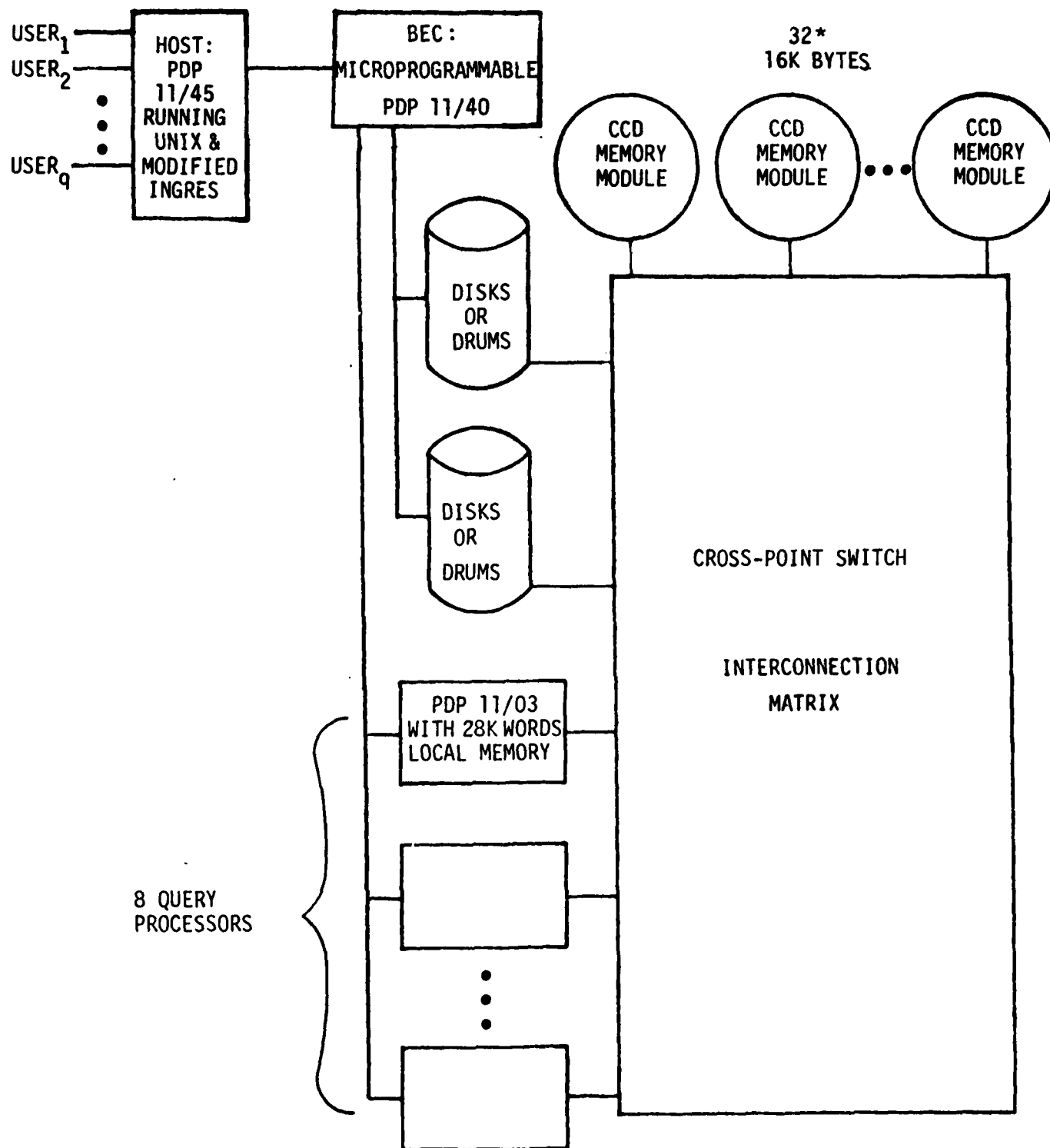


Figure 8-3: Proposed DIRECT Physical Configuration

January 1980

9 Summary

The systems we have described illustrate the spectrum of approaches that can be used to provide support for data management in a distributed environment. They were chosen to demonstrate the panorama of choices that are available to Navy designers concerned with providing data management facilities for shipboard systems.

The systems described encompass logical architectures ranging from very loosely coupled designs oriented to geographically distributed systems to tightly coupled designs dependent on multiprocessors or special architectures. They range in complexity from very simple designs oriented to the commercial marketplace to highly complex systems with sophisticated algorithms that are being developed in academic and research institutions. A summary of how the systems studied compare in architecture and complexity is shown in Figure 9-1.

There seem to be three main approaches to providing distributed data base management services:

- o the network data manager approach
- o the integrated DDBMS approach
- o the loosely coupled "let the application do the work" approach

The network data manager approach divides distributed data base management into global (coordination) and local (back-end data management) parts and concerns itself only with the global (coordination) part. Some research at the National Bureau of Standards [Kimbleton 79] is trying to establish the feasibility of this approach for systems that will support heterogeneous back-end data base managers. (There are difficult as yet unsolved problems related to translation of data structure and commands.) CCA used an approach amenable to this type of division in SDD-1, but with only one type of data manager, their own Datacomputer. They could thus take advantage of the special characteristics of their system to provide better performance.

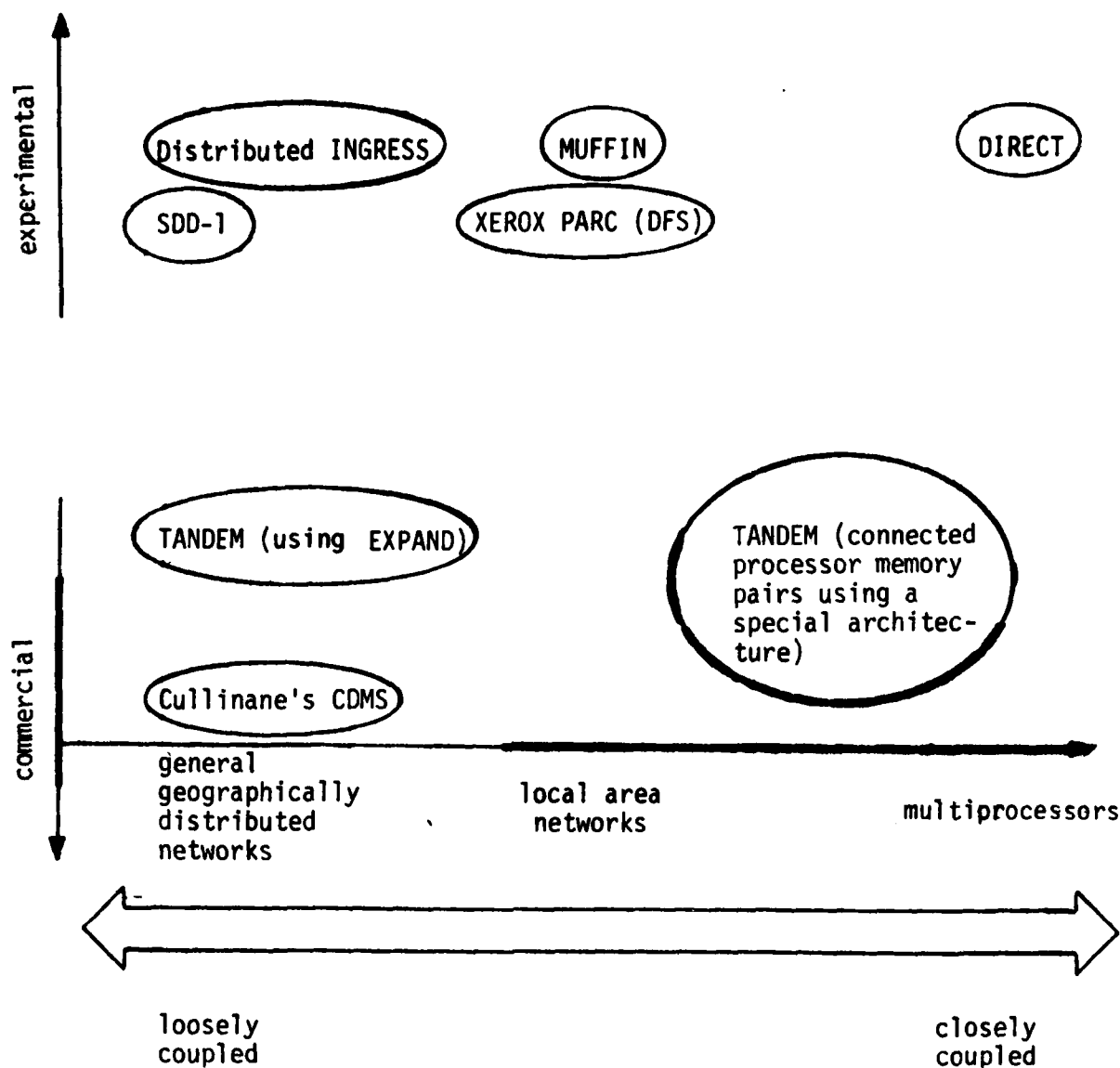


Figure 9-1: The Network Environment of Some Distributed Systems

INGRES, MUFFIN and DIRECT are examples of integrated systems that do not seem to be as amenable to this type of division. However, they probably gain in performance through the use of an integrated design.

Xerox's DFS system is partitioned in a somewhat different way that leaves traditional data base management functions such as formatting and structuring of data for the user controlled front-ends.

Cullinane's CDMS and TANDEM's ENSCRIBE are examples of systems where only the basic facilities for reliable data access in a network are provided by the system. A network data manager such as that described above could be implemented on top of the existing software to provide additional services. A summary of how the systems we looked at fit into this pattern is shown in Figure 9-2.

Another issue approached differently in the systems studied is how to divide functionality between front and back-end components when a user is logged onto one machine and the data he is accessing is on another. A few systems (Cullinane's CDMS, DIRECT) put most of the functionality in the back-end; some others (TANDEM, Xerox) put most in the front-end or leave it to the user to supply; the integrated systems take an intermediate approach that allows for the possibility of provision of services by a third or several intermediate machines. This spectrum of approaches is illustrated in Figure 9-3.

There are several functional areas of particular importance in distributed system architecture. The approaches taken by the systems we looked at tend to characterize their architecture and provide a rudimentary taxonomy for modeling these systems.

These areas are:

- o transparency of location.
- o approach to redundant data,
- o consistency control mechanisms:
 - o to ensure that transactions are treated as indivisible (atomic) entities (intratransaction consistency),

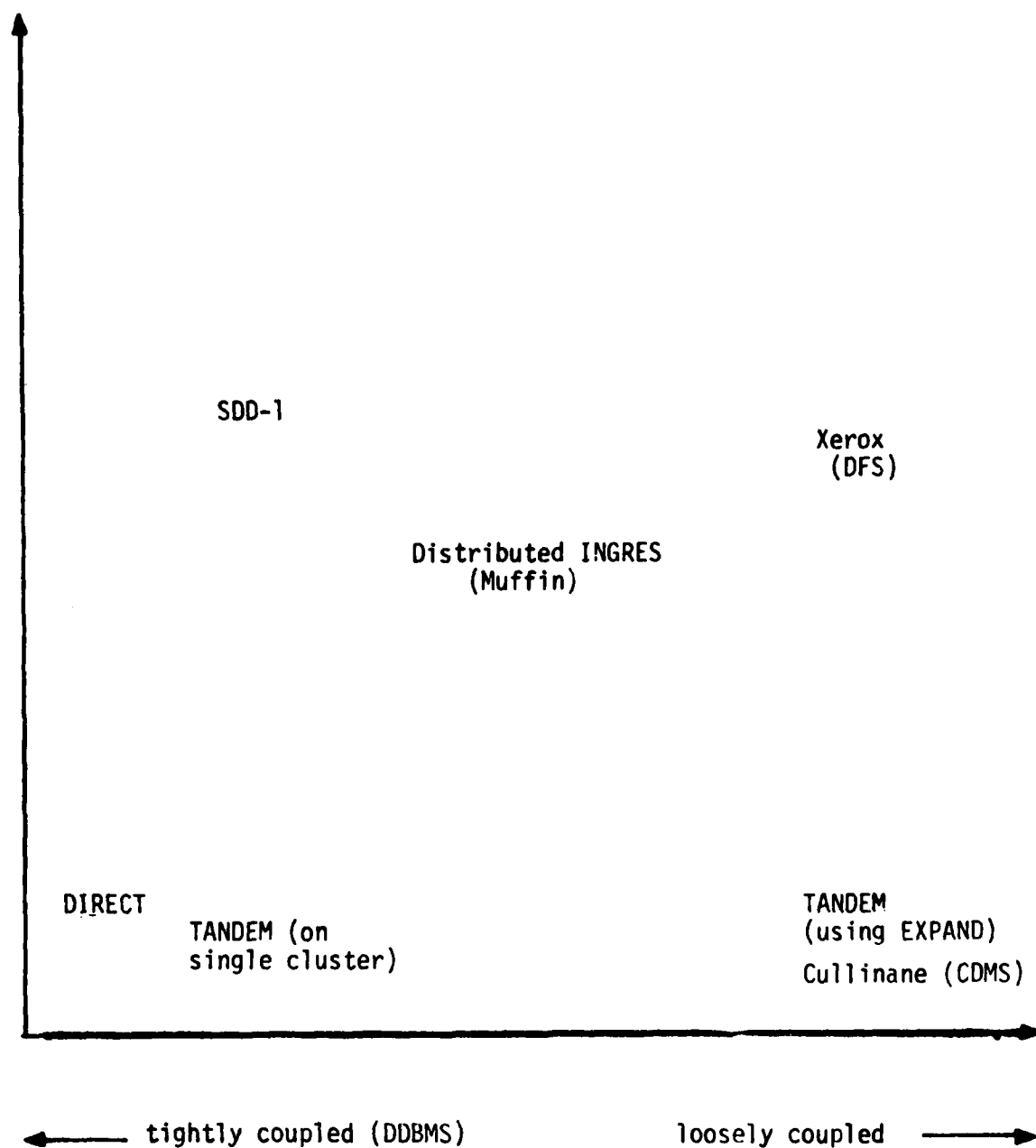


Figure 9-2: Coupling between Logical Components in Some Distributed Systems

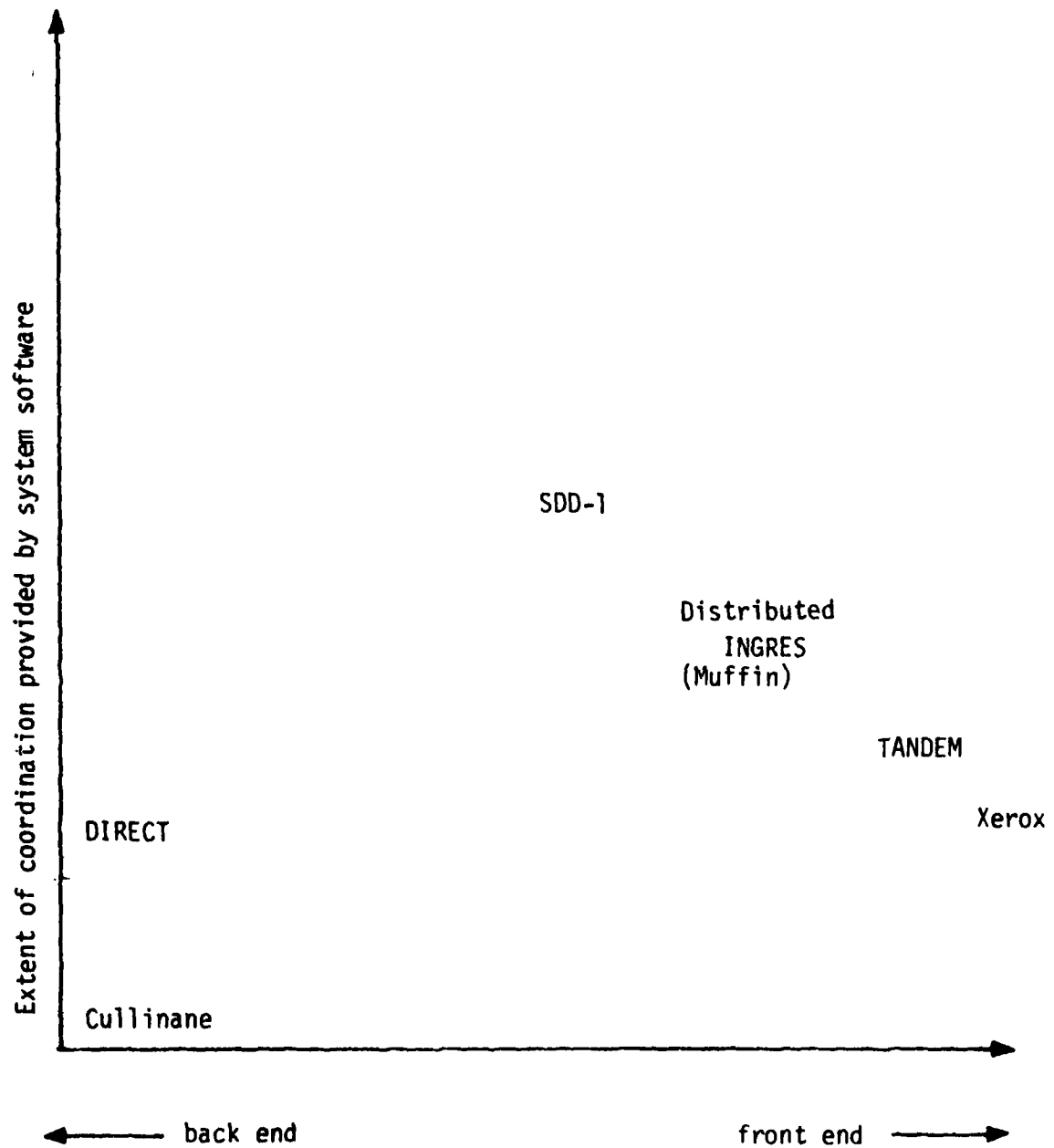


Figure 9-3: Division of DBMS Functionality among System Components

	Cullinane	Xerox
TRANSPARENCY OF LOCATION	none (location information will eventually be provided in their IDD)	none(file identifiers that are unique across the network, i.e. which implicitly specify the host, are used)
HANDLING OF REDUNDANT COPIES OF DATA	by user	by user, but DFS will guarantee atomicness of transactions
CONSISTENCY CONTROLS; SERIALIZABILITY		locking
ATOMICNESS (OF TRANSACTIONS)		intentions list
APPROACH TO CONTROL	partitioned (independent DBMSs responsible for own data)	one "coordinator" is selected to control a transaction through completion
RELIABILITY MECHANISMS		intentions lists; stable storage; basic file structure that updates all data associated with a transaction before changing pointers

Figure 9-4: Comparison of Functional Issues

Tandem	SDD-1	Distributed INGRESS/MUFFIN	DIRECT
only one copy of the data base manager ENSCRIBE is run per cluster; thus it functions like a local data base manager; in general networks (accessed using EXPAND), the user must explicitly specify location	distributed global directories	partitioned global directory	global directory
mirrored discs; any redundancy at data item level must be handled by the user	automatically by system	automatically by system	
locking	conflict graph analysis	locking plus directing updates through a primary site	
through redundancy of system architecture plus checkpointing mechanism	variation of two-phase commit	variation of two-phase commit	
embedded in global operating system	distributed; a transaction module handles a given class of transactions	primary site model	single back end machine
dual paths plus two-fold redundancy of all components at the physical level; mirrored discs plus software to control this embedded in the NonStop version of their operating system; check-pointing between copies of a program on different machines	spoolers; two-phase commit	variation of two-phase commit	

Figure 9-5: Comparison of Functional Issues, Continued

- o to ensure that a group of transactions executed concurrently are equivalent to the same group of transactions executed in some serial order (intertransaction consistency);
- o resiliency mechanisms that increase the reliability of the distributed systems. Generally the resiliency mechanisms are designed to recover from:
 - o failure of the node generating or processing a transaction (e.g. sending updates),
 - o failure of one or more nodes maintaining copies of data to be updated (e.g. receiving updates),
 - o various problems with the underlying communication network that prevent access to certain pieces of the data base.

A summary of how the systems we looked at approach these functional issues is shown in Figure 9-4.

REFERENCES

[Alsberg 76]

Alsberg, P.A., and Day, J.D.
A Principle for Resilient Sharing of Distributed Resources.
 Technical Report, University of Illinois, Urbana, Illinois,
 1976.

[Bernstein 78]

Bernstein, Philip, Rothnie, James, Goodman, Nathan, and
 Papadimitriou, Christos.
 The Concurrency Control Mechanism of SDD-1: A System for
 Distributed Databases (The Fully Redundant Case).
IEEE Transactions on Software Engineering SE-4(3), May,
 1978.

[Datacomputer 78]

Datacomputer Version 5 User Manual
 Computer Corporation of America, 1978.

[DeWitt 79]

DeWitt, David J.
 DIRECT - A Multiprocessor Organization for Supporting
 Relational Database Management Systems.
IEEE Transactions on Computers C-28(6), June, 1979.

[Gray 78]

Gray, J.N.
Lecture Notes in Computer Science. Volume 60: Notes on Data
Base Operating Systems.
 Springer Verlag, 1978, .

[Held 75]

Held, G., Stonebraker, M., and Wong, E.
 NCC.
 In NCC. AFIPS Press, Montvale, NJ, 1975.

[Israel 78]

Israel, Jay E., Mitchell, James G., and Sturgis, Howard E.
Separating Data from Function in a Distributed File System.
 Technical Report CSL-78-5, XEROX PARC, September, 1978.

[KARP 66]

Karp, R. M. and Miller, R. E.
 Properties of a Model for Parallel Computation:
 Determinacy, Termination, Queueing.
Society of Industrial and Applied Mathematics (14),
 December, 1966.

[Kimbleton 79]

Kimbleton, Stephen; Wang, Pearl; and Fong, Elizabeth.
 XNDM: An Experimental Network Data Manager.
 In Proceedings of the Fourth Berkeley Conference on
Distributed Data Management and Computer Networks.
 Lawrence Berkeley Laboratory, University of California ,
 Berkeley, August, 1979.

[Metcalf, R. M., and Boggs, D.R. 76]

Metcalf, R. M., and Boggs, D.R.

ETHERNET: distributed packet switching for local computer networks.

Communications of the ACM (19), July, 1976.

[Paxton 79]

Paxton, William H.

A Client-Based Transaction System to Maintain Data Integrity.

In Proceedings of the Seventh Symposium on Operating System Principles, Pacific Grove, California. ACM (SIGOPS), December, 1979.

[Rothnie 77]

Rothnie, James B., and Goodman, Nathan.

An Overview of the Preliminary Design of SDD-1: A system for Distributed Databases.

In Proceedings of the Second Berkeley Workshop on Distributed Data Management and Computer Networks. Lawrence Berkeley Laboratory, University of California, Berkeley, May, 1977.

[Rothnie 79]

Rothnie, James B., Jr., Bernstein, Philip A., Fox, Stephen A., Goodman, Nathan, Hammer, Michael M., Landers, Terry A., Reeve, Christopher, L. Shipman, David W., and Wong, Eugene. SDD-1: A system for Distributed Databases.

Technical Report CCA-02-79, CCA, August, 1979.

[Rothnie 80]

Rothnie, James B., Jr.

Distributed DBMS no longer just a concept.

Data Communications, January, 1980.

[Rowe 79]

Rowe, Lawrence A., and Birman, Kenneth P.

Network Support for a Distributed Data Base System.

In Proceedings of the Fourth Berkeley Conference on Distributed Data Management and Computer Networks.

Lawrence Berkeley Laboratory, University of California, Berkeley, August, 1979.

[Stonebraker 76]

Stonebraker, Michael, Wong, Eugene, Kreps, Peter, and Held, Gerald.

The Design and Implementation of INGRES.

Transactions on Database Systems (3), September, 1976.

[Stonebraker 77]

Stonebraker, Michael.

A Distributed Data Base Version of INGRES.

In Proceedings of the Second Berkeley Workshop on Distributed Data Management and Computer Networks.

Lawrence Berkeley Laboratory, University of California, Berkeley, August, 1977.

[Stonebraker 78]

Stonebraker, Michael.
Concurrency Control and Consistency of Multiple Copies of
Data in Distributed Ingress.
In Proceedings of the Third Berkeley Workshop on Distributed
Data Management and Computer Networks. Lawrence Berkeley
Laboratory, University of California, Berkeley, 1978.
reprinted in IEEE Transactions on Software Engineering, Vol.
SE-5, No.3, May 1979.

[Stonebraker 79]

Stonebraker, Michael.
MUFFIN: A Distributed Data Base Machine.
In Proceedings of the First International Conference on
Distributed Computing Systems, Huntsville, Alabama. IEEE
Computer Society, October, 1979.

[Swinehart 79]

Swinehart, Daniel; McDaniel, Gene; and Boggs, David.
WFS: A Simple Shared File System for a Distributed
Environment.
In Proceedings of the Seventh Symposium on Operating System
Principles, Pacific Grove, California. ACM (SIGOPS),
December, 1979.

[Thomas 79]

Thomas, R. H.
A Majority Consensus Approach to Concurrency Control for
Multiple Copy Databases.
Transactions on Database Systems 4(2), June, 1979.

[Wong 77]

Wong, Eugene.
Retrieving Dispersed Data from SDD-1: A System for
Distributed Databases.
In Proceedings of the Second Berkeley Workshop on
Distributed Data Management and Computer Networks.
Lawrence Berkeley Laboratory, University of California,
Berkeley, May, 1977.

January 1980

Index

A-cell 43
Access planning 30, 34
Alsberg 39
Alto 11
ARPANET 27, 44
Associative memory 49
Atomic 39
Atomic property 16, 33, 36
Atomicness 13, 27

Back-end 11, 37, 44
Back-end controller 49, 52
Backup processor 20
BEC 49, 52
Bernstein 34
Boggs 11

CCA 27
CDMS 7
Client 11
Client-server approach 11
CODASYL 7
Commercial 19
Computer Corporation of America 27
Concurrency control 30, 39
Concurrent updates 33, 52
Conflict graph analysis 27, 33
Consistency 13
Contention based network 11
Coordinator 16
Cross-point switch 49
Cullinane 7
Cullinane Data Management System 7

D-cell 43
Data base machine 43
Data flow graphs 35
Data Module 30
Datacomputer 28
Datalanguage 28
Deadlock detection 39
DeWitt 49
DFS 11, 13
DIRECT 49
Directory information 27
Distributed File System 11, 13
Distributed INGRES 37
Distributed query processing 30, 34
DM 30
DMA link 49
Dynabus 19

ENSCRIBE 19, 20
Ethernet 11

QUEL 43
 Query packet 49
 Query processor 49

 Read phase 32
 Read-set 32
 Redundancy 19
 Redundant copies 39
 Redundant data 27
 Relational algebra 49
 Relational data model 28, 37, 49
 Reliability enhancements 27
 Reliable Network 27, 30
 Reliable writing 35
 RELNET 27, 30
 Rothnie 27
 Rowe 41, 44

 SDD-1 27
 Serializability 13, 33
 Server 11
 Site monitoring 32
 Slave INGRES 37, 44
 SNOOP 39
 Spooled files 28
 Spooler 35, 41
 Stable storage 16
 Stonebraker 37, 43
 Sturgis 13
 Swinehart 11
 Synchronization 33

 Tandem 19
 Thomas 36
 Three phase division 27
 Timestamps 28, 34, 36
 TMs 30
 Transaction control 32, 35
 Transaction Module 27, 30
 Transaction Modules 30
 Transaction oriented system 13
 Transaction processing 27
 Transparency 27
 Two-phase commit 28, 36, 39

 UNIX 37

 VAX 37, 44

 Wait for graph 41
 WFS 11
 Wong 34
 Worker 16
 Worker list 16
 Write phase 33
 Write-set 32

 Xerox 11
 Xerox PARC 11

4-
DT